Estimation ensembliste par analyse par intervalles Application à la localisation d'un véhicule

Michel Kieffer

ORSAY
n° d'ordre : 5633

UNIVERSITE DE PARIS-SUD
CENTRE SCIENTIFIQUE D'ORSAY

THÈSE DE DOCTORAT

Spécialité : Automatique

par

Michel KIEFFER

Titre :

Estimation ensembliste par analyse par intervalles Application à la localisation d'un véhicule

Soutenue le 18 janvier 1999 devant la commission d'examen

MM.	Pierre BERTRAND	Président
	Raja CHATILA	Rapporteur
	Jürgen GARLOFF	Rapporteur
	Luc JAULIN	
	Dominique MEIZEL	
	Éric WALTER	

Remerciements

Je tiens à exprimer ma reconnaissance à Monsieur Guy DEMOMENT, directeur du Laboratoire des Signaux et Systèmes, pour m'avoir accueilli au sein de cet établissement pendant la durée de cette thèse.

Je voudrais témoigner toute ma gratitude à Monsieur Pierre BERTRAND, qui a su me donner des conseils utiles pour le choix de ce sujet de thèse, et le remercier chaleureusement d'avoir lu avec intérêt mon travail et accepté de présider mon jury de thèse.

Mes deux rapporteurs, Messieurs Raja CHATILA et Jürgen GARLOFF, ont chacun accepté, malgré de nombreuses responsabilités administratives, de consacrer une partie de leur temps (pris peut-être sur leurs vacances de Noël) à une lecture approfondie de ce manuscrit. Je les en remercie vivement.

Je suis très reconnaissant à Monsieur Dominique MEIZEL pour l'intérêt qu'il a porté à mon travail, et surtout pour sa disponibilité lorsqu'il a fallu répondre à mes questions sur les aspects appliqués de cette thèse.

La participation de Monsieur Luc JAULIN à l'encadrement de ce travail a été déterminante. Que ce soit lors de mes séjours à Angers ou plus récemment pendant son détachement au Laboratoire des Signaux et Systèmes, les idées nombreuses qu'il a su me communiquer ainsi que ses remarques quant au contenu et à la présentation de ce document ont été essentielles.

Pendant ces trois années Monsieur Eric WALTER, mon directeur de thèse, a su orienter efficacement mon travail tout en me laissant une liberté très appréciable. Très disponible pendant cette préparation, il n'a ménagé ni ses efforts, ni son temps. Je lui en suis très reconnaissant.

Je voudrais exprimer toute ma gratitude au personnel administratif et technique du laboratoire et de Supélec, aux chercheurs permanents et plus particulièrement aux thésards et anciens thésards qui ont su créer une ambiance conviviale très agréable au sein du laboratoire.

J'ai également une pensée émue pour mes étudiants de Marne-la-Vallée, pour mes acteurs du club théâtre de l'E.N.S. de Cachan ainsi que pour les membres de l'Orchestre et Choeur des Universités de Paris ; tous m'ont donné tant d'occasions de regarder un peu au-delà de mes intervalles.

Je voudrai encore remercier mes parents, ma famille, ainsi que tous mes amis qui m'ont encouragé et soutenu. Ce travail leur est dédié.

Un dernier petit mot pour Hélène, simplement pour la remercier de m'offrir sa présence, son sourire, son affection...

omnia autem probate
quod bonum est tenete
ab omni specie mala abstinete vos

1 Th 5 21-22

Table des matières

Chapitre 1

Introduction

> *Kourou, le 4 juin 1996. Le vol inaugural d'Ariane 5 s'est soldé par un échec. Environ* 40 *secondes seulement après le démarrage de la séquence de vol, le lanceur, qui se trouvait alors à une altitude de quelques* 3700 *mètres, a dévié de sa trajectoire, s'est brisé et a explosé.*[1]

Cet échec a retardé de près d'un an le programme Ariane 5, et coûté plusieurs centaines de millions d'euros aux Etats participant au programme spatial européen, sans compter le préjudice porté à l'image de l'agence spatiale européenne.

La commission d'enquête indépendante nommée quelques jours après l'échec du vol a en révélé l'origine dans son communiqué de presse du 19 juillet 1996. L'explosion serait la conséquence d'un dysfonctionnement du logiciel intégré à un capteur. La conversion d'un nombre réel en un entier n'aurait pu s'effectuer correctement parce que le nombre réel était trop grand pour être représenté par un entier standard.

> *L'exception [l'erreur du] logiciel interne du SRI [système de référence inertielle] s'est produite pendant une conversion de données de représentation flottante à 64 bits en valeurs entières à 16 bits. Le nombre en représentation flottante qui a été converti avait une valeur qui était supérieure à ce que pouvait exprimer un nombre entier à 16 bits. Il en est résulté une erreur d'opérande. Les instructions de conversion de données (en code Ada) n'étaient pas protégées contre le déclenchement d'une erreur d'opérande bien que d'autres conversions de variables comparables présentes à la même place dans le code aient été protégées.*[1]

Le logiciel incriminé fonctionnait parfaitement sur les fusées Ariane 4 ; sa mise en œuvre sur Ariane 5, sans tenir compte des spécificités du nouveau lanceur, a eu des conséquences désastreuses.

Cette expérience met en évidence l'une des faiblesses majeures de la réalisation de calculs sur ordinateurs, à savoir le manque de fiabilité des résultats fournis, même quand les équations mises en œuvre sont correctes dans leur principe. En effet, un ordinateur est contraint de manipuler des nombres représentés

[1] Extraits du communiqué de presse du 19 juillet 1996 (`http: //www.cnes.fr/espace_pro/news/rapport_501.html`)

avec une précision finie, c'est-à-dire par un nombre de *bits* limité (par exemple 16 pour les nombre entiers, 64 pour les réels en virgule flottante, etc.). Ainsi, seul un nombre limité de réels peut être représenté de manière exacte : avec 64 bits, il est possible de représenter au maximum 2^{64} réels différents. Les autres ne sont qu'approchés. Lorsqu'un petit nombre d'opérations est effectué sur ces nombres approchés, le résultat final est en général très proche de celui qui serait obtenu si on disposait d'une représentation exacte (avec une précision infinie), par contre, lorsque des millions d'opérations sont effectuées, les incertitudes s'accumulant, le résultat final peut tout à fait être très éloigné du résultat exact. Cet état de fait n'est cependant pas une fatalité, il est parfaitement possible d'utiliser une représentation des réels en utilisant un nombre beaucoup plus élevé de bits, qui sera beaucoup plus lourde à manipuler par un ordinateur et inutile dans la plupart des cas. Le tout est d'identifier les programmes où une grande précision de représentation des données numériques sera nécessaire.

Parmi les recommandations faites par la commission d'enquête pour la poursuite du programme Ariane 5, le $5^{ème}$ point a plus particulièrement retenu notre attention.

> *5. Revoir tous les logiciels de vol (y compris les logiciels intégrés), et en particulier :*
> *(...)*
> *- Vérifier la plage des valeurs prises dans les logiciels par l'une quelconque des variables internes ou de communication.*[2]

Cette disposition aurait sans doute permis d'éviter l'erreur qui a conduit à l'échec du vol. En connaissant la plage de variation de la variable interne x incriminée, il aurait été possible de détecter que pour certaines valeurs de x, il n'était pas possible de réaliser une conversion vers un entier à 16 bits.

Cette préoccupation de déterminer les plages de variation de variables, jointe à un souci d'évaluer la précision avec laquelle les résultats de calculs étaient obtenus, a été à l'origine, il y a près de 40 ans de l'*analyse par intervalles* aux Etats-Unis. R. E. Moore, dès 1959, donne une méthode systématique et *garantie* d'obtention des plages des valeurs prises par les variables d'un algorithme. Pour mieux comprendre la suite de cette introduction et le contenu de cette thèse, définissons tout d'abord la notion d'algorithme.

Un algorithme est un ensemble d'opérations effectuées séquentiellement et faisant intervenir un certain nombre de variables dites d'*entrée*, qui sont fournies par l'utilisateur, par des instruments de mesure ou qui sont le résultat d'autres algorithmes. Ces manipulations permettent d'obtenir un ou plusieurs résultats correspondant aux variables de *sortie*. Mathématiquement, ces deux dernières phrases peuvent être résumées par

$$(y_1, y_2, \ldots, y_m) = \mathsf{f}\,(x_1, x_2, \ldots, x_n)\,,$$

où $(x_1, x_2, \ldots, x_n)$ désigne les n variables d'entrée, $(y_1, y_2, \ldots, y_m)$ correspond aux m variables de sortie

[2] Extrait du communiqué de presse du 19 juillet 1996 (`http: //www.cnes.fr/espace_pro/news/rapport_501.html`)

et f représente l'algorithme de calcul. Le nombre des variables d'entrée et de sortie peut varier de un à quelques millions.

Le problème auquel était confronté l'analyse par intervalles était d'évaluer l'ensemble décrit par les variables de sortie d'un algorithme lorsque les variables d'entrées varient (généralement dans des ensembles ou des *intervalles* donnés *a priori*, voir la figure 1-1).

Pour l'algorithme mis en jeu dans le SRI de la fusée Ariane 5, un certain nombre de variables d'entrée (accélération de la pesanteur, vitesse de rotation de la terre, accélération au départ de la fusée, etc.) interviennent dans l'algorithme. Ces grandeurs ne sont connues qu'avec une précision finie, tributaire des dispositifs de mesure utilisés. Par contre, il est possible de définir des intervalles contenant de manière garantie les valeurs de ces variables d'entrée. Il s'agit alors, comme le recommande la commission d'enquête, de déterminer le domaine de variation possible des variables intermédiaires de l'algorithme ainsi que des variables de sorties afin de vérifier qu'aucune n'entraîne de comportement anormal du système global.

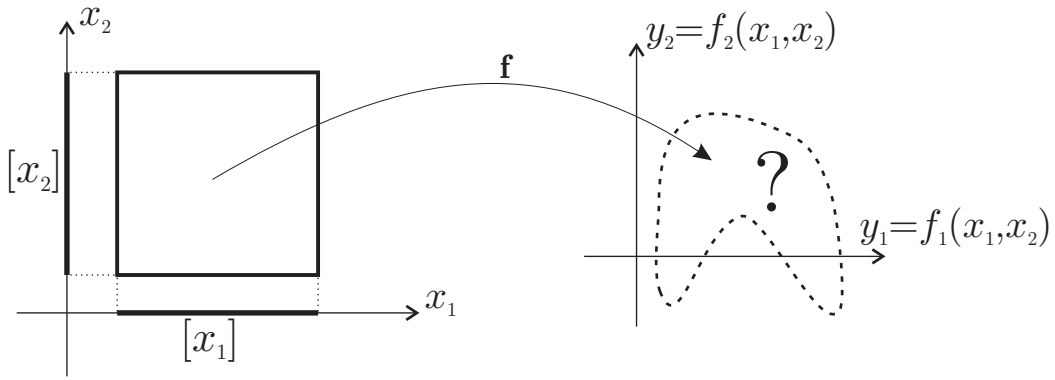


Figure 1-1: Problème-type de détermination de l'ensemble décrit par les sorties (y_1, y_2) lorsqu'on connaît les intervalles $[x_1]$ et $[x_2]$ de variation des variables d'entrée x_1 et x_2 de l'algorithme f.

L'analyse par intervalles permet de caractériser ces domaines de variation de manière approchée, mais garantie, c'est-à-dire que si le domaine *réel* d'une variable de sortie est l'intervalle $[-4, 2]$, les méthodes utilisant l'analyse par intervalles fourniront *toujours* un encadrement *extérieur* du domaine de variation par exemple $[-5, 3]$, et *jamais* $[-3, 1]$.

L'analyse par intervalles s'est révélée fournir des solutions originales à des problèmes mathématiques standards tels que la recherche des solutions d'un système d'équations linéaires ou non linéaires ou l'optimisation de fonctions coût non nécessairement unimodales. L'originalité de l'approche réside dans l'aspect garanti des résultats obtenus. Par exemple, pour la minimisation d'une fonction coût, un encadrement de *tous* les arguments du minimum *global* sont obtenus.

Toutefois, l'impact de ces techniques est resté relativement faible auprès des scientifiques et des industriels, à cause des difficultés de mise en œuvre. Ce problème est en cours de résolution depuis le début des années 90, grâce à l'apparition de bibliothèques mathématiques dédiées à l'analyse par intervalles

et écrites dans des langages élaborés tels ADA, C++ (C-XSC ou PROFIL/BIAS) ou Fortran90 (INTLIB 90). La *surcharge* d'opérateurs, c'est-à-dire la possibilité par exemple de redéfinir le comportement de l'addition, de la soustraction et des fonctions usuelles pour des intervalles, permet de manipuler ces objets aussi aisément que les entiers ou les réels en virgule flottante. Il n'est alors plus nécessaire de récrire les programmes afin d'obtenir un encadrement des variables de sortie, les variables intervalles sont manipulées de manière totalement transparente par l'utilisateur.

Actuellement, le développement de ces techniques et le nombre de domaines dans lesquelles elles sont appliquées sont en forte croissance. En témoignent les articles présentés à des conférences récentes sur l'analyse par intervalles, par exemple à l'*International Workshop on Application of Interval Computation* APIC'1995 à El Paso, INTERVAL'96 à Würzburg, INTERVAL'98 à Nanjin, *etc.*. Les domaines d'application vont de la physique à l'économie en passant par la robotique. Un journal dédié à l'analyse par intervalles, *Reliable Computing*, est publié par Kluwer. Les analystes par intervalles se sont également positionnés sur le réseau Internet, créant à l'initiative de R. B. Kearfott un site entièrement dédié à l'analyse par intervalles dont l'adresse est

`http: //cs.utep.edu/interval-comp/main.html`

Ce site constitue une mine d'informations tant sur les méthodes que sur les domaines d'application. Une *mailing list*, permet également un dialogue actif entre les chercheurs du monde entier.

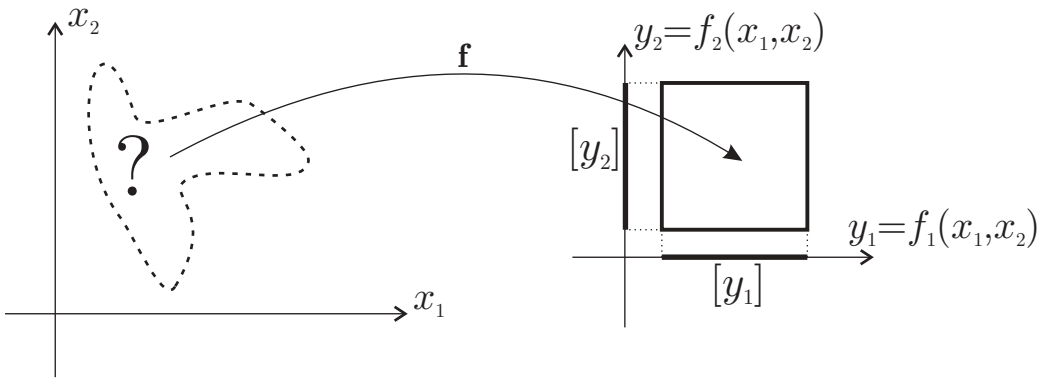


Figure 1-2: Problème-type d'inversion ensembliste : détermination de l'ensemble décrit par les entrées (x_1, x_2) sachant que les sorties y_1 et y_2 de l'algorithme f doivent rester dans les intervalles $[y_1]$ et $[y_2]$.

Au sein du Laboratoire des Signaux et Systèmes, l'intérêt pour l'analyse par intervalles en automatique et traitement du signal a débuté en 1992, lors de la thèse de Luc Jaulin sous la direction d'Éric Walter. Il y a été en particulier mis en évidence que de nombreux problèmes d'automatique s'expriment non pas comme une recherche des ensembles décrits par les variables de sortie $y_1, \ldots, y_m$ d'un algorithme f lorsque ses entrées $x_1, \ldots, x_n$ varient, mais comme la recherche des valeurs possibles des entrées d'un algorithme sachant que ses sorties appartiennent à un ensemble connu *a priori*. On parle de problèmes d'*inversion*

ensembliste : il s'agit de trouver l'antécédent d'un ensemble par un algorithme donné (voir la figure 1-2).

Ainsi, le problème de l'*identification* des paramètres d'un modèle dans un contexte d'erreurs bornées peut s'exprimer sous cette forme. D'autres problèmes, tel la caractérisation de domaines de stabilité de systèmes, la synthèse de correcteurs robustes, etc. peuvent également se formuler de cette manière et être résolus de façon approchée à l'aide de l'analyse par intervalles.

Un algorithme d'inversion ensembliste utilisant l'analyse par intervalles a été développé à l'occasion de la thèse de Luc Jaulin ; il permet de résoudre de manière approchée mais garantie un problème d'inversion ensembliste lorsque l'ensemble à inverser est un *pavé* (produit cartésien d'intervalles) ou lorsqu'il est décrit par des inégalités. Cet algorithme fournit des solutions approchées à ces différents problèmes. L'ensemble des valeurs possibles de $x_1, \ldots, x_n$ est encadré par une union de pavés *contenant* l'ensemble antécédent de l'ensemble à inverser (voir la figure 1-3).

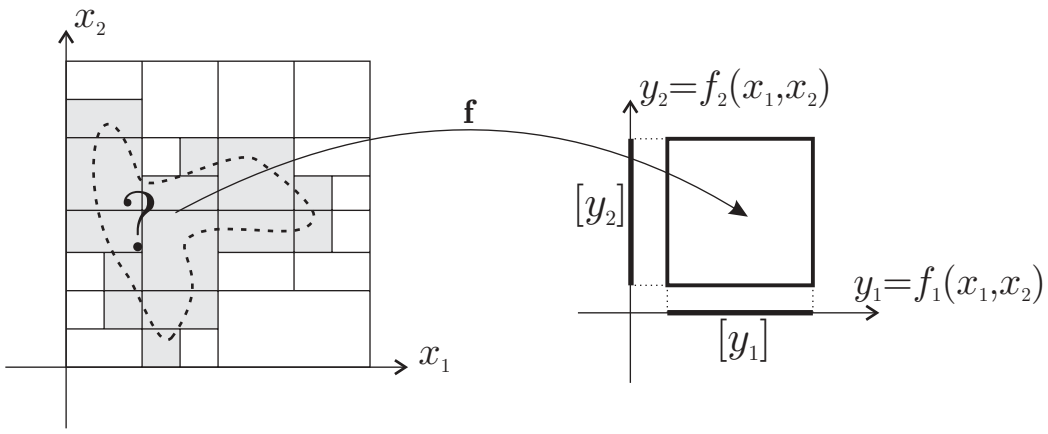


Figure 1-3: Ensemble approché (en gris) de l'ensemble des valeurs possibles des entrées (délimité par des pointillés) obtenu par inversion ensembliste utilisant l'analyse par intervalles.

Dans sa thèse défendue en 1997, Olivier Didrit est allé au-delà de la démonstration de faisabilité effectuée dans la thèse précédente, en traitant des problèmes de plus grande complexité.

L'objectif de mon travail a été de continuer à démontrer l'applicabilité des techniques intervalles à des problèmes d'automatique originaux, en mettant l'accent en particulier sur l'estimation récursive qui n'avait pratiquement pas été abordée dans le contexte de l'analyse par intervalles.

Ce document est donc centré sur l'estimation. Dans une première partie, l'estimation des paramètres d'un modèle est abordée. Dans une seconde partie, nous traiterons de l'estimation d'état récursive. Un soin particulier a été apporté au développement de nouvelles structures et de nouveaux algorithmes. Un lecteur peu soucieux de programmation en C++ peut parfaitement choisir d'ignorer les parties traitant de ces aspects. Cependant, la mise en œuvre informatique représentant une part importante de ce travail (quelques 20000 lignes de code ont été écrites) et les algorithmes développés n'ayant de sens qu'implantés sur ordinateur, il nous a paru utile d'insister sur cet aspect qui présente ses propres difficultés.

Le chapitre 2 présentera les outils de l'analyse par intervalles employés tout au long de ce mémoire. Si certains sont très classiques, d'autres ont été introduits ou utilisés intensivement pour la première fois, comme par exemple la notion de test booléen, l'introduction de masques permettant d'accélérer les algorithmes d'inversion ensembliste, la fonction *et-q-relaxé*, qui constituera un outil puissant pour le traitement de données aberrantes ou encore la notion de fonction d'inclusion multivaluée, qui se révèlera très efficace pour encadrer le domaine de variation de fonctions discontinues.

Deux parties constituent le chapitre 3, consacré à l'identification des paramètres de systèmes non-linéaires. Lorsque l'erreur entre mesures et sorties prédites par le modèle est considérée comme aléatoire à distribution gaussienne, une identification par minimisation d'une fonction coût quadratique est envisagée. L'algorithme de Hansen permet alors de trouver tous les arguments du minimum global de cette fonction coût. Par contre, lorsque l'erreur doit rester bornée, une méthode par inversion ensembliste utilisant l'analyse par intervalles permet de fournir toutes les valeurs des paramètres compatibles avec les mesures et les hypothèses faites sur les erreurs. Ces deux méthodes d'identification seront présentées et illustrées par des exemples.

La localisation d'un robot sera abordée au chapitre 4 dans un contexte statique. Un robot est placé dans un environnement connu dont il dispose d'une carte des principaux éléments. Des capteurs à ultrasons (ou tout autre dispositif télémétrique) lui fournissent des mesures des distances aux plus proches obstacles dans des directions données. Disposant de ces informations, le robot doit déterminer sa position dans son environnement. Ce problème sera résolu dans le contexte d'erreurs de mesure bornées. Plusieurs exemples illustratifs montreront les propriétés de l'algorithme d'estimation proposé (robustesse à l'égard de données aberrantes, possibilité de prendre en compte les ambiguïtés éventuelles de données).

Au chapitre 5 seront présentées les principales contributions de cette thèse. Les solutions fournies par les algorithmes d'inversion ensembliste sont constituées d'une union de pavés qu'il était difficile de manipuler à cause de leur manque d'organisation. La notion de sous-pavage représenté par des arbres binaires est introduite afin de structurer ces ensembles. Il est alors possible de réaliser l'inversion ensembliste non plus seulement de pavés ou d'ensembles décrits par des inégalités, mais de n'importe quel ensemble borné qui puisse être contenu dans une union de pavés. En outre, l'inversion ensembliste récursive peut être réalisée. Un algorithme récursif d'estimation d'état utilisant ces outils est proposé. Cet algorithme nécessite en outre une étape de calcul de l'image directe d'un ensemble par une fonction. Deux méthodes fournissant un encadrement extérieur de l'ensemble image avec une précision arbitraire sont proposées, l'une d'entre elles est originale. La convergence de cette dernière est établie. Enfin, un exemple illustratif simple de suivi d'une balle rebondissant est abordé.

L'exemple du robot du chapitre 4 est ensuite repris au chapitre 6 dans un contexte évolutif. En utilisant l'algorithme d'estimation d'état du chapitre précédent, il devient possible de réaliser une poursuite du robot mobile dans son environnement. La localisation se fait toujours à l'aide de capteurs ultrasonores. Lors de l'évolution, l'ensemble des positions possibles du robot est prédit à chaque instant puis corrigé lorsque de nouvelles mesures sont disponibles. Les propriétés de robustesse vis-à-vis de mesures aberrantes

et la possibilité de tenir compte des ambiguïtés seront présentées sur un exemple.

Ce travail de thèse ne correspond pas au point final d'une activité de recherche. Après un rappel des principaux résultats obtenus, le chapitre 7 donnera quelques pistes de recherche ultérieures concernant l'analyse par intervalles en automatique en général et en robotique en particulier.

Chapitre 2

Notions d'analyse par intervalles

2.1 Introduction

Ce chapitre commencera par un rappel des notions de base de l'analyse par intervalles. Un intervalle peut être à la fois considéré comme un ensemble de réels et comme le représentant d'un réel incertain. De ce fait, quelques opérations ensemblistes ainsi que l'extension aux intervalles de l'arithmétique sur les réels seront rappelées. Une extension aux vecteurs et matrices d'intervalles sera brièvement introduite. Par la suite, nous verrons comment il est possible d'évaluer une fonction sur des intervalles, et d'obtenir des encadrements plus ou moins précis de l'ensemble des valeurs prises par une fonction sur un intervalle grâce aux fonctions d'inclusion. L'évaluation de la qualité de ces fonctions sera possible grâce à l'introduction d'une norme sur les ensembles compacts. Le cas de fonctions d'inclusion spéciales, pour des algorithmes faisant intervenir des tests, sera également envisagé.

Pour plus de détails, consulter plutôt les ouvrages de Moore [Moore79], Hansen [Hansen92], et Neumaier [Neumaier90] pour une présentation générale de l'analyse par intervalle et celui de Klatte *et al.* [Klatte et al.93] pour une approche plus informatique. Nous adopterons les notations de ce dernier ouvrage.

Dans une seconde partie, un aperçu de l'implantation informatique en C++ des différentes structures (intervalles, vecteurs d'intervalles) sera présentée. Trois ouvrages ont été particulièrement utiles pour la mise en œuvre des algorithmes développés ; [Metairie et al.90] ou [Delannoy91] constituent une bonne introduction en français au C++ et aux spécificités de ce langage, en particulier par rapport au C. [Lippman89] est beaucoup plus complet ; les problèmes assez épineux de gestion dynamique de la mémoire, par exemple, y sont relativement bien présentés.

2.2 Intervalles - définitions et opérations élémentaires

Un intervalle réel est un sous-ensemble fermé borné connexe de $\mathbf{R}$. On le note

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbf{R} | \underline{x} \leq x \leq \overline{x}\}, \tag{2.1}$$

où $\underline{x}$ et $\overline{x}$ sont respectivement ses bornes inférieure et supérieure. L'ensemble des intervalles réels sera noté $\mathbf{IR}$. Plus généralement, si $\mathcal{D}$ est un ensemble de réels, $\mathbf{I}\mathcal{D}$ désigne l'ensemble des intervalles de $\mathcal{D}$ que l'on peut construire à partir de $\mathcal{D}$ et contenant *uniquement* ses éléments.

Un intervalle est dit *dégénéré* si $\underline{x} = \overline{x}$; dans ces conditions, $[x] \in \mathbf{R} \subset \mathbf{IR}$.

Les intervalles étant des ensembles, les notions d'égalité ($=$), d'appartenance ($\in$), d'inclusion stricte ($\subset$) et large ($\subseteq$), ainsi que d'intersection ($\cap$) sont parfaitement définies. Cependant, la réunion de deux intervalles n'étant en général pas un intervalle, l'*union convexe* de deux intervalles ($\underline{\cup}$) est introduite ; elle correspond au plus petit intervalle contenant l'union de ces deux intervalles

$$[x] \underline{\cup} [y] = \left[\min(\underline{x}, \underline{y}), \max(\overline{x}, \overline{y})\right]. \tag{2.2}$$

Pour caractériser un intervalle, on peut également utiliser

- son *centre* ou *milieu* : $m([x]) = \frac{\overline{x}+\underline{x}}{2}$,

et

- sa *longueur* ou son *diamètre* : $w([x]) = \overline{x} - \underline{x}$,

ou

- son *rayon* : $r([x]) = \frac{\overline{x}-\underline{x}}{2}$.

En outre, pour un intervalle $[x]$ donné, sa plus petite et sa plus grande valeur absolue (respectivement $\langle [x] \rangle$ et $|[x]|$) sont définies de la manière suivante

$$\langle [x] \rangle = \min\{|x|, x \in [x]\} = \min\{0, |\underline{x}|, |\overline{x}|\}, \tag{2.3}$$

$$|[x]| = \max\{|x|, x \in [x]\} = \max\{|\underline{x}|, |\overline{x}|\}. \tag{2.4}$$

L'ensemble de ces notions est illustré sur la figure 2-1

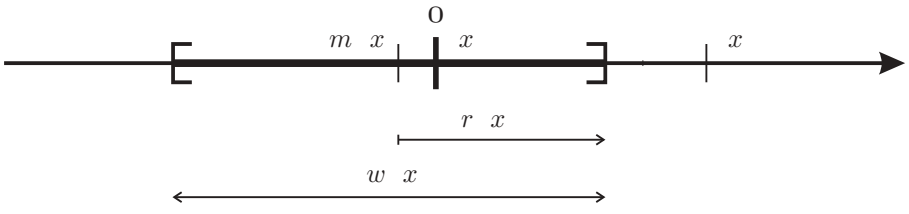


Figure 2-1: Caractéristiques d'un intervalle réel.

Enfin, la longueur *relative* est définie de la manière suivante :

$$\begin{aligned} w_{\text{rel}}([x]) &= \frac{w([x])}{\langle [x] \rangle} \text{ si } 0 \notin [x] \\ &= w([x]) \text{ sinon.} \end{aligned} \tag{2.5}$$

De manière classique, un intervalle correspond à un ensemble de réels, mais il peut également représenter l'ensemble dans lequel peut se trouver un réel incertain. Ainsi, le diamètre relatif peut être interprété comme un majorant de l'erreur relative commise en considérant un élément quelconque $x \in [x]$, lorsque $0 \notin [x]$.

Exemple 2.2.1 *On mesure une température $\theta = 21\ ^\circ C$. Le fabriquant du thermomètre garantit une mesure à $\pm 1\ ^\circ C$. Par conséquent, la vraie valeur de la température se trouve dans l'intervalle $[\theta] = [20\ ^\circ C, 22\ ^\circ C]$. Ainsi,*

$$w_{rel}([20\ ^\circ C, 22\ ^\circ C]) = \frac{2}{20} = 10\%$$

sera donc l'erreur relative maximale que l'on pourra commettre sur la température. ♢

Etendre aux intervalles les opérations arithmétiques de base $\{+, -, \times, /\}$ est une idée naturelle pour évaluer l'ensemble que décrit la somme, le produit, *etc.* de deux réels incertains inclus dans des intervalles. Cet ensemble image est défini de la manière suivante

$$\text{soit } \circ \in \{+, -, \times, /\}, \ [x] \circ [y] = \{x \circ y | x \in [x] \text{ et } y \in [y]\}. \tag{2.6}$$

La caractérisation de $[x] \circ [y]$ se fait à l'aide du formulaire

$$\begin{cases} [x] + [y] = \left[\underline{x} + \underline{y}, \overline{x} + \overline{y}\right], \\ [x] - [y] = \left[\underline{x} - \overline{y}, \overline{x} - \underline{y}\right], \\ [x] \times [y] = \left[\min\left(\underline{x}.\underline{y}, \overline{x}.\underline{y}, \underline{x}.\overline{y}, \overline{x}.\overline{y}\right), \max\left(\underline{x}.\underline{y}, \overline{x}.\underline{y}, \underline{x}.\overline{y}, \overline{x}.\overline{y}\right)\right], \\ [x] / [y] = [x] \times \left[1/\overline{y}, 1/\underline{y}\right], \text{ si } 0 \notin [y] \text{ et indéfini sinon.} \end{cases} \tag{2.7}$$

Remarquons que la distributivité de la multiplication par rapport à l'addition n'est plus respectée. On a ainsi

$$[a] \times ([b] + [c]) \subseteq [a] \times [b] + [a] \times [c], \tag{2.8}$$

c'est la propriété de *sous-distributivité*. Par contre les propriétés de commutativité et d'associativité restent valables.

Ces opérations élémentaires sont *monotones pour l'inclusion*, c'est-à-dire que

$$[x] \subseteq [x'] \text{ et } [y] \subseteq [y'] \Rightarrow [x] \circ [y] \subseteq [x'] \circ [y']\,. \tag{2.9}$$

2.3 Intervalles étendus

La division telle qu'elle a été définie précédemment ne permet pas de considérer le cas d'un intervalle diviseur contenant 0. Pour résoudre cette difficulté, il convient d'introduire la notion d'*intervalle étendu*. Pour cela, l'ensemble des réels est complété en lui ajoutant $+\infty$ et $-\infty$ pour obtenir

$$\overline{\mathbf{IR}} = \mathbf{IR} \cup \{[-\infty, r]\,|r \in \mathbf{R}\} \cup \{[l, +\infty]\,|l \in \mathbf{R}\} \cup \{[-\infty, +\infty]\}\,. \tag{2.10}$$

Les intervalles étendus permettent de compléter la définition de la division en considérant le cas où 0 appartient à l'intervalle diviseur

$$[x] \,/\, [y] = \begin{cases} [-\infty, +\infty] & \text{si } \underline{x} < 0 < \overline{x} \text{ ou } [x] = 0 \text{ et } [y] = 0, \\ \left[\overline{x}/\underline{y}, +\infty\right] & \text{si } \overline{x} \leq 0 \text{ et } \underline{y} < \overline{y} = 0, \\ [-\infty, \overline{x}/\overline{y}] \cup \left[\overline{x}/\underline{y}, +\infty\right] & \text{si } \overline{x} \leq 0 \text{ et } \underline{y} < 0 < \overline{y}, \\ [-\infty, \overline{x}/\overline{y}] & \text{si } \overline{x} \leq 0 \text{ et } 0 = \underline{y} < \overline{y}, \\ \left[-\infty, \underline{x}/\underline{y}\right] & \text{si } 0 \leq \underline{x} \text{ et } \underline{y} < \overline{y} = 0, \\ \left[-\infty, \underline{x}/\underline{y}\right] \cup [\underline{x}/\overline{y}, +\infty] & \text{si } 0 \leq \underline{x} \text{ et } \underline{y} < 0 < \overline{y}, \\ [\underline{x}/\overline{y}, +\infty] & \text{si } 0 \leq \underline{x} \text{ et } 0 = \underline{y} < \overline{y}. \end{cases}$$

Remarquons que, dans deux cas, nous obtenons la réunion de deux intervalles, ce qui nécessitera un traitement algorithmique particulier (voir également le paragraphe 2.6)

2.4 Vecteurs et matrices d'intervalles

L'ensemble des vecteurs d'intervalles réels (ou *pavés*) de dimension n est noté $\mathbf{IR}^n$. Un élément $[\mathsf{a}]$ de cet ensemble sera écrit

$$[\mathsf{a}] = \begin{pmatrix} [a]_1 \\ \ldots \\ [a]_n \end{pmatrix}.$$

L'ensemble des matrices d'intervalles réels de dimension $n \times m$ est noté $\mathbf{IR}^{n\times m}$.

$$[\mathsf{A}] = \begin{pmatrix} [a]_{11} & \cdots & [a]_{1m} \\ \vdots & \ddots & \vdots \\ [a]_{n1} & \cdots & [a]_{nm} \end{pmatrix},$$

est un élément de cet ensemble.

Les notions ensemblistes sont étendues composantes par composantes ; ainsi, dans le cas de l'inclusion, soient $[\mathsf{a}] \in \mathbf{IR}^n$ et $[\mathsf{b}] \in \mathbf{IR}^n$,

$$[\mathsf{a}] \subset [\mathsf{b}] \Leftrightarrow [\mathsf{a}]_i \subset [\mathsf{b}]_i \text{ pour tout } i = 1, \ldots, n.$$

La définition du centre, de la longueur et de la longueur relative d'un vecteur ou d'une matrice sont également étendus composante par composante. En outre, pour les vecteurs, on introduit les notions de longueur maximale w_∞, de longueur pondérée maximale $w_{\mathrm{p}\infty}$ et de longueur relative maximale $w_{\mathrm{rel}\infty}$: soit $[\mathsf{a}] \in \mathbf{IR}^n$ et $\mathsf{p} = (p_1, \ldots, p_n)^{\mathrm{T}} \in (\mathbf{R}^+)^n$

$$w_\infty([\mathsf{a}]) = \max_{i=1,\ldots,n} w([\mathsf{a}]_i), \tag{2.11}$$

$$w_{\mathrm{p}\infty}([\mathsf{a}], \mathsf{p}) = \max_{i=1,\ldots,n} w([\mathsf{a}]_i \times p_i), \tag{2.12}$$

$$w_{\mathrm{rel}\infty}([\mathsf{a}]) = \max_{i=1,\ldots,n} w_{\mathrm{rel}}([\mathsf{a}]_i). \tag{2.13}$$

2.5 Fonctions d'inclusion

Soit f une fonction réelle d'une variable réelle définie sur un domaine $\mathcal{D} \subset \mathbf{R}$. La définition de cette fonction est étendue aux variables intervalles $[x] \subset \mathcal{D}$ en considérant l'*évaluation intervalle* de f sur $[x]$.

$$f([x]) = \{f(x) \,|\, x \in [x]\}. \tag{2.14}$$

C'est un ensemble qui contient toutes les valeurs prises par f sur $[x]$. Ce n'est évidemment pas toujours un intervalle, mais par construction, l'évaluation intervalle est toujours monotone pour l'inclusion :

$$[x] \subseteq [y] \Rightarrow f([x]) \subseteq f([y]). \tag{2.15}$$

Il est aisé de caractériser l'évaluation intervalle des fonctions monotones usuelles. Ainsi, par exemple

$$\begin{aligned}
[x]^2 &= \left[\langle [x]\rangle^2, |[x]|^2\right], \\
\sqrt{[x]} &= \left[\sqrt{\underline{x}}, \sqrt{\overline{x}}\right], \text{ si } \underline{x} \geq 0, \\
\exp([x]) &= [\exp(\underline{x}), \exp(\overline{x})], \\
\tan([x]) &= [\tan(\underline{x}), \tan(\overline{x})], \text{ si } [x] \subseteq [-\pi/2, \pi/2].
\end{aligned}$$

Les fonctions trigonométriques sinus et cosinus, n'étant pas monotones, leur évaluation intervalle est plus délicate car il faut considérer différents cas.

Exemple 2.5.1 *Notons* $[y] = \sin([x])$. *Excluons le cas simple* $w([x]) \geq 2\pi$, *pour lequel* $[y] = [-1, 1]$. *Il existe* $k \in \mathbf{Z}$ *tel que* $\underline{x_1} = \underline{x} - 2k\pi \in [0, 2\pi]$. *Soit* $\overline{x_1} = \underline{x_1} + d([x])$. *L'intervalle image de* $[x_1] = \left[\underline{x_1}, \overline{x_1}\right]$ *sera le même que celui de* $[x]$ *à cause de la périodicité de la fonction sinus, et on montre facilement que*

$$\begin{array}{lll}
\textit{si } \frac{3\pi}{2} \in [x_1] \textit{ ou } \frac{7\pi}{2} \in [x_1], & \underline{y} = -1, \\
\quad \textit{sinon,} & \underline{y} = \inf(\sin \underline{x_1}, \sin \overline{x_1}), \\
\textit{si } \frac{\pi}{2} \in [x_1] \textit{ ou } \frac{5\pi}{2} \in [x_1], & \overline{y} = 1, \\
\quad \textit{sinon,} & \overline{y} = \sup(\sin \underline{x_1}, \sin \overline{x_1}).
\end{array}$$

♢

L'évaluation intervalle d'une fonction f plus générale est souvent beaucoup plus difficile. La recherche du minimum et du maximum de f sur un intervalle $[x]$ représente en effet un véritable problème d'optimisation en soi. Il est par contre en général facile de définir une nouvelle fonction *intervalle*, aisée à évaluer, dont les images contiendront les évaluations intervalles de f.

Définition 2.5.1 *Soit* $f : \mathcal{D} \subset \mathbf{R} \to \mathbf{R}$, *la fonction* $f_{[]} : \mathbf{I}\mathcal{D} \to \mathbf{IR}$ *est une* fonction d'inclusion *[Ratschek et al.88] de* f *si pour tout* $[x] \in \mathbf{I}\mathcal{D}$ *elle vérifie*

$$f([x]) \subseteq f_{[]}([x]). \tag{2.16}$$

Cette fonction d'inclusion sera dite convergente *lorsque la propriété suivante est satisfaite :*

$$\textit{si } w([x]) \to 0 \textit{ alors } w\left(f_{[]}([x])\right) \to 0. \tag{2.17}$$

♢

Insistons sur le fait que $f_{[]}([x])$ est bien un intervalle. Les fonctions d'inclusion sont définies de la même manière pour des fonctions ayant pour valeur des vecteurs ou des matrices de réels. L'un des objectifs de l'analyse par intervalles est d'obtenir une fonction d'inclusion donnant des intervalles les plus proches

possibles des évaluations intervalles fournies pour la fonction réelle correspondante. Lorsque l'inclusion (2.16) est une égalité, la fonction $f_{[]}$ est dite *minimale*.

Il existe plusieurs méthodes pour obtenir une fonction d'inclusion de la fonction f. La plus simple est de remplacer les occurrences des variables scalaires $(x, y, \dots)$ par les variables intervalles correspondantes $([x], [y], \dots)$, une fonction d'inclusion *naturelle* de f est alors obtenue. Cependant, bien souvent, cette fonction d'inclusion n'est pas la meilleure (elle n'est minimale que lorsque chacune des variables n'intervient qu'une seule fois dans l'expression de f) ; en outre, la taille des intervalles obtenus dépend assez fortement de l'expression initiale de la fonction.

Exemple 2.5.2 *Comparons les performances des fonctions d'inclusion établies à partir des trois formulation de la même fonction suivantes*

$$\begin{aligned} f_1(x) &= x(x+1), \\ f_2(x) &= x \times x + x, \\ f_3(x) &= x^2 + x, \\ f_4(x) &= (x + \tfrac{1}{2})^2 - \tfrac{1}{4}. \end{aligned}$$

Pour l'intervalle $[x] = [-1, 1]$*, les évaluations intervalle des expressions précédentes sont*

$$\begin{aligned} f_{1[]}([x]) &= [x]\,([x]+1) = [-2, 2]\,, \\ f_{2[]}([x]) &= [x] \times [x] + [x] = [-2, 2]\,, \\ f_{3[]}([x]) &= [x]^2 + [x] = [-1, 2]\,, \\ f_{4[]}([x]) &= \left([x] + \tfrac{1}{2}\right)^2 - \tfrac{1}{4} = \left[-\tfrac{1}{4}, 2\right]. \end{aligned}$$

Selon l'expression de la fonction, l'encadrement obtenu sera donc plus ou moins grossier (voir figure.2-2). Les deux expressions $[x]\times[x]$ *et* $[x]^2$ *ne sont pas équivalentes, dans le premier cas, chacune des occurences de* x *peut varier indépendemment. Précisons que l'ensemble image de la fonction initiale est* $\left[-\frac{1}{4}, 2\right]$*.* $f_{4[]}$ *est donc minimale.* ♢

Une fonction d'inclusion qui fournit un intervalle image qui n'est pas égal à l'évaluation intervalle de la fonction réelle sur l'intervalle considéré est dite *pessimiste*.

Il n'existe pas de méthode systématique permettant pour une fonction donnée d'obtenir une fonction d'inclusion minimale. Cependant, on peut constater que plus une variable apparaîtra fréquemment, plus le pessimisme des intervalles images sera important. Cela peut s'interpréter par le fait que deux occurrences d'une même variable sont considérées comme variant indépendamment l'une de l'autre. Ainsi, les incertitudes se combinent, pour donner un résultat plus mauvais que si l'on avait considéré les occurrences comme variant identiquement.

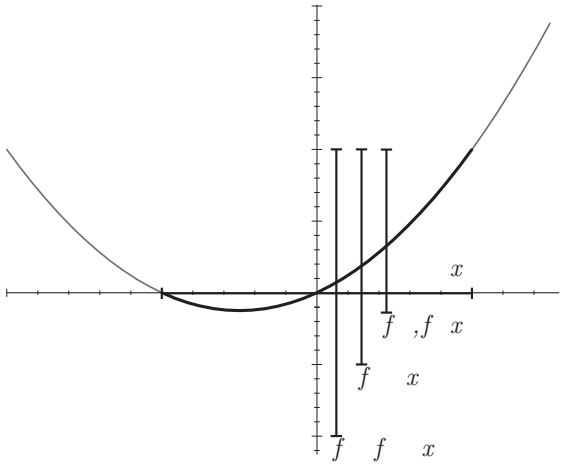


Figure 2-2: Comparaison de fonctions d'inclusion d'une même fonction.

Il existe d'autres types de fonctions d'inclusion, faisant intervenir des développements en série de la fonction initiale. Ainsi, pour une fonction $f : \mathcal{D} \longrightarrow \mathbf{R}$, une fois dérivable sur un intervalle $[x] \subset \mathcal{D}$, on sait que $\forall x, m \in [x]\,,\ \exists \xi \in [x]$ tel que

$$f(x) = f(m) + (x - m)\, f'(\xi)\,. \tag{2.18}$$

Par conséquent

$$f(x) \in f(m) + (x - m)\, f'([x])\,, \tag{2.19}$$

d'où

$$f([x]) \subseteq f(m) + ([x] - m)\, f'_{[]}([x])\,. \tag{2.20}$$

La fonction intervalle définie par

$$f_{[],m}([x]) = f(m) + ([x] - m)\, f'_{[]}([x])\,, \tag{2.21}$$

avec $m \in [x]$ et $f'_{[]}$ une fonction d'inclusion de la dérivée de f, est donc une fonction d'inclusion de f. Cette fonction est appelée *forme centrée standard* de f sur l'intervalle $[x]$, avec pour centre m. Elle donne de bons résultats, son pessimisme est assez faible lorsque l'intervalle considéré a une faible longueur.

Exemple 2.5.3 *Considérons la fonction* $f(x) = x^2 \exp(x) - x \exp\left(x^2\right)$. *Le tableau comparatif ci-dessous permet de comparer les encadrements fournis par les deux types de fonction d'inclusion présentés (fonction d'inclusion naturelle* $f_{[]}$ *et forme centrée en* 1 $f_{[],1}$*) avec l'évaluation intervalle de la fonction. Les*

résultats sont donnés avec 4 *chiffres significatifs.*

$[x]$	$f([x])$	$f_{[]}([x])$	$f_{[],1}([x])$
$[0.5, 1.5]$	$[-4.148, 0]$	$[-13.82, 9.44]$	$[-25.07, 25.07]$
$[0.9, 1.1]$	$[-0.05380, 0]$	$[-1.697, 1.612]$	$[-0.5050, 0.5050]$
$[0.99, 1.01]$	$[-0.0004192, 0]$	$[-0.1636, 0.1628]$	$[-0.004656, 0.004656]$

On constate que pour des intervalles de grande taille, la forme centrée peut s'avérer plus pessimiste que la fonction d'inclusion naturelle, par contre elle est assez efficace pour des intervalles de petite taille (voir par exemple [Moore79]). ♢

Pour évaluer la qualité d'une fonction d'inclusion, il est nécessaire d'évaluer la distance entre l'évaluation fournie par la fonction d'inclusion et l'évaluation intervalle de la fonction sur un intervalle test.

2.6 Norme, convergence

La distance entre intervalles (et plus généralement entre sous-ensembles compacts de $\mathbf{R}^n$) dont nous allons parler dans ce paragraphe sera utile pour les démonstrations de convergence des algorithmes du chapitre 5.

Comme nous nous intéressons ici principalement à des intervalles ou à des pavés, il convient d'utiliser une norme qui leur soit adaptée. Dans le cas de la norme Euclidienne, la boule est une sphère. Par contre, lorsqu'on considère la norme L_∞ sur $\mathbf{R}^n$, les distances sont définies de la manière suivante

$$\text{soit } \mathsf{a} \in \mathbf{R}^n \text{ et } \mathsf{b} \in \mathbf{R}^n,\ d_\infty(\mathsf{a}, \mathsf{b}) = \max_{i=1,\ldots,n} \{|a_i - b_i|\}. \tag{2.22}$$

Avec cette norme, une boule de rayon R est un hypercube de côté $2R$ (voir figure 2-3).

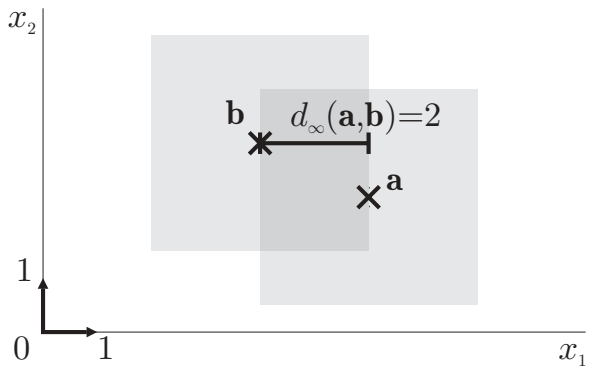


Figure 2-3: Norme L_∞ et distance entre 2 points.

La distance peut s'interpréter en termes de rayon de boule : la distance entre a et b est le rayon de la plus petite boule centrée en a et contenant b ou réciproquement. Par contre, tenter de définir une distance d'un point a à un ensemble compact $\mathcal{B}$ est un peu plus compliqué. Deux choix peuvent être faits : soit on considère le rayon de la plus petite boule centrée en a et contenant entièrement $\mathcal{B}$, soit on considère chacun des points b de $\mathcal{B}$, on évalue le rayon de la plus petite boule centrée en b contenant a et on prend le plus petit de ces rayons. Comme la première solution a l'inconvénient de fournir des valeurs qui dépendent fortement de la forme de $\mathcal{B}$, la seconde solution est adoptée. La pseudo-distance ainsi construite est notée $d_0(\mathsf{a}, \mathcal{B})$; elle correspond à une sorte de gonflement de $\mathcal{B}$ pour que $\mathcal{B}$ gonflé contienne a (voir figure 2-4). Ce n'est pas une distance, car lorsque $\mathsf{a} \in \mathcal{B}$, $d_0(\mathsf{a}, \mathcal{B}) = 0$, même si $\mathsf{a} \neq \mathcal{B}$.

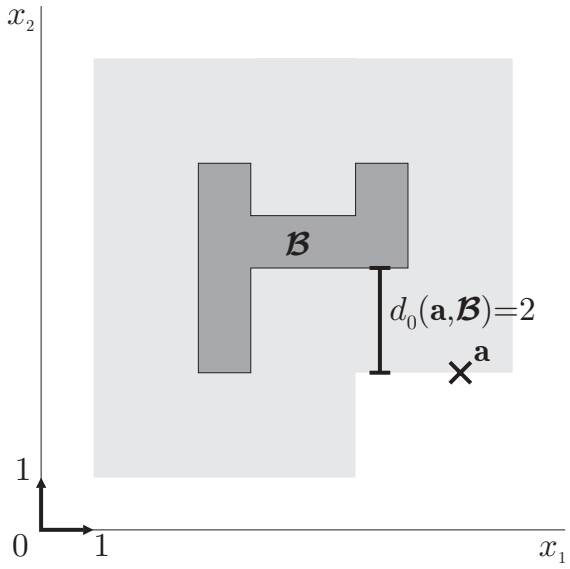


Figure 2-4: Calcul de la distance d'un point a à un ensemble compact $\mathcal{B}$ (en gris foncé).

Pour évaluer la distance entre deux ensembles compacts $\mathcal{A}$ et $\mathcal{B}$, on pourra reprendre l'idée précédente, en considérant un point a de $\mathcal{A}$ et en évaluant $d_0(\mathsf{a}, \mathcal{B})$, puis en définissant la distance de $\mathcal{A}$ à $\mathcal{B}$ comme la plus grande de ces pseudo-distances lorsque a parcours $\mathcal{A}$. On note alors $d_0(\mathcal{A}, \mathcal{B}) = \max_{\mathsf{a} \in \mathcal{A}} d_0(\mathsf{a}, \mathcal{B})$. Pour interpréter graphiquement $d_0(\mathcal{A}, \mathcal{B})$, on peut reprendre l'idée d'un gonflement de $\mathcal{B}$ qui doit cette fois contenir entièrement $\mathcal{A}$. Cependant, $d_0(.,.)$ n'est en fait pas une distance car elle s'annule lorsque $\mathcal{A} \subset \mathcal{B}$. On construit alors la *distance de Hausdorff* $d(\mathcal{A}, \mathcal{B})$ en considérant la plus grande des deux quantités $d_0(\mathcal{A}, \mathcal{B})$ et $d_0(\mathcal{B}, \mathcal{A})$. La définition suivante résume cette construction.

Définition 2.6.1 *Soient $\mathcal{A}$ et $\mathcal{B}$ deux sous-ensembles compacts de $\mathbf{R}^n$. La* distance de Hausdorff $d(\mathcal{A}, \mathcal{B})$

utilisant la norme L_∞ est

$$\begin{aligned} & d(\mathcal{A},\mathcal{B}) = \max\left\{d_0(\mathcal{A},\mathcal{B}), d_0(\mathcal{B},\mathcal{A})\right\}, \\ & \text{où } d_0(\mathcal{A},\mathcal{B}) = \max_{\mathsf{a}\in\mathcal{A}} d_0(\mathsf{a},\mathcal{B}), \\ & \text{avec } d_0(\mathsf{a},\mathcal{B}) = \min_{\mathsf{b}\in\mathcal{B}} d_\infty(\mathsf{a},\mathsf{b}) \text{ et } d_\infty(\mathsf{a},\mathsf{b}) = \max_{i=1,\ldots,n}\left\{|a_i - b_i|\right\}. \end{aligned} \tag{2.23}$$

♢

Proposition 1 *[Berger79b] Les distances de Hausdorff telles que $d(.,.)$ constituent une norme pour l'ensemble de compacts de $\mathbf{R}^n$.* ♢

Les propriétés suivantes de pseudo-distance d_0 et de distance de Hausdorff seront utiles, en particulier au chapitre 3

Soient $\mathcal{A}$, $\mathcal{B}$, et $\mathcal{C}$ des sous-ensembles compacts de $\mathbf{R}^n$ et $[\mathsf{x}]$, un pavé de $\mathbf{R}^n$ on a alors les propriétés suivantes

Lemme 2.6.1 *si $\mathcal{B}\subset\mathcal{C}$ alors $d_0(\mathcal{B},\mathcal{C}) = 0$ et $d_0(\mathcal{A},\mathcal{C}) \leq d_0(\mathcal{A},\mathcal{B})$;*

Lemme 2.6.2 *si $\mathcal{B}\subset[\mathsf{x}]$ et $w_\infty([\mathsf{x}]) \leq \epsilon$ alors $d(\mathcal{B},[\mathsf{x}]) \leq \epsilon$;*

Lemme 2.6.3 *si $w_\infty([\mathsf{x}]) \leq \epsilon$ et $d_0([\mathsf{x}],\mathcal{A}) > \epsilon$ alors $[\mathsf{x}]\cap\mathcal{A} = \emptyset$.* ♢

Preuve :

1. Soit $\mathsf{b}\in\mathcal{B}$. $d_0(\mathsf{b},\mathcal{C}) = \min_{\mathsf{c}\in\mathcal{C}} d_\infty(\mathsf{b},\mathsf{c})$, or $\mathcal{B}\subset\mathcal{C}$ implique $\min_{\mathsf{c}\in\mathcal{C}} d_\infty(\mathsf{b},\mathsf{c}) = 0$ donc $d_0(\mathcal{B},\mathcal{C}) = 0$.
   Soit $\mathsf{a}\in\mathcal{A}$. $\mathcal{B}\subset\mathcal{C}$ entraîne $\min_{\mathsf{c}\in\mathcal{C}} d_\infty(\mathsf{a},\mathsf{c}) \leq \min_{\mathsf{b}\in\mathcal{B}} d_\infty(\mathsf{a},\mathsf{b})$, par conséquent $d_0(\mathcal{A},\mathcal{C}) \leq d_0(\mathcal{A},\mathcal{B})$.

2. $\mathcal{B}\subset[\mathsf{x}]$, donc d'après la première propriété, $d_0(\mathcal{B},[\mathsf{x}]) = 0$. Comme $w_\infty([\mathsf{x}]) \leq \epsilon$ et $\mathcal{B}\subset[\mathsf{x}]$, pour tout $\mathsf{b}\in\mathcal{B}$ et pour tout $\mathsf{x}\in[\mathsf{x}]$, $d_\infty(\mathsf{x},\mathsf{b}) \leq \epsilon$. par conséquent, $d_0(\mathsf{x},\mathcal{B}) \leq \epsilon$ et $d_0([\mathsf{x}],\mathcal{B}) \leq \epsilon$ donc $d([\mathsf{x}],\mathcal{B}) \leq \epsilon$.

3. (Par l'absurde) Supposons qu'il existe $\mathsf{x}_0 \in [\mathsf{x}]\cap\mathcal{A}$. Soit $\mathsf{x}\in[\mathsf{x}]$, $d_0(\mathsf{x},\mathcal{A}) = \min_{\mathsf{a}\in\mathcal{A}} d_\infty(\mathsf{x},\mathsf{a}) \leq d_\infty(\mathsf{x},\mathsf{x}_0) \leq \epsilon$ car $w_\infty([\mathsf{x}]) \leq \epsilon$. Par conséquent, $d_0([\mathsf{x}],\mathcal{A}) \leq \epsilon$, ce qui vient contredire l'une des hypothèses. ■

Dans le cas unidimentionnel, on peut montrer que la distance de Hausdorff s'exprime pour des intervalles de la manière suivante

$$d([x],[y]) = \max\left\{\left|\underline{x}-\underline{y}\right|, \left|\overline{x}-\overline{y}\right|\right\}. \tag{2.24}$$

Il est alors possible de reprendre l'exemple 2.5.3 pour comparer les fonctions d'inclusion à l'aide de la distance de Hausdorff.

Exemple 2.6.1 *Reprenons la fonction* $f(x) = x^2 \exp(x) - x \exp\left(x^2\right)$. *Le tableau ci-dessous permet de comparer les performances de la fonction d'inclusion naturelle* $f_{[]}$ *et de la forme centrée en* 1, $f_{[],1}$ *à l'aide de la distance de Hausdorff des intervalles résultats avec les intervalles fournis par* $f([x])$. *Les résultats sont donnés avec* 4 *chiffres significatifs.*

$[x]$	$f([x])$	$d\left(f([x]), f_{[]}([x])\right)$	$d\left(f([x]), f_{[],1}([x])\right)$
$[0.5, 1.5]$	$[-4.148, 0]$	9.672	25.07
$[0.9, 1.1]$	$[-0.05380, 0]$	1.6432	0.5050
$[0.99, 1.01]$	$[-0.0004192, 0]$	0.1632	0.004656

Les deux dernières lignes du tableau permettent de comparer les vitesse de convergence des deux fonctions d'inclusion : lorsqu'on réduit la taille de l'intervalle initial d'un facteur 10, *la distance entre évaluation intervalle et intervalle fourni par la fonction d'inclusion naturelle est également diminuée d'un facteur* 10, *pour la forme centrée, ce rapport est de plus de* 100. ♢

Cette propriété est exploitée par les algorithmes intervalles de type *branch-and-bound*. Lorsqu'une fonction d'inclusion donne un résultat trop pessimiste sur un intervalle ou un pavé donné, en coupant ce pavé en deux *sous-intervalles* ou *sous-pavés* et en appliquant la fonction d'inclusion sur chacune des deux parties, l'image obtenue sera moins pessimiste que l'image initiale, pour peu que la fonction d'inclusion soit convergente. Les propriétés de convergence d'une classe particulière de fonctions d'inclusion ont été étudiées par Hansen [Han69] et Moore [Moore79]. Elles ne concernent que les fonctions d'une variable vectorielle à valeur scalaire. Une extension aux fonctions à valeurs vectorielles sera présentée au chapitre 5.

Définition 2.6.2 *Une fonction d'inclusion* $\mathsf{f}_{[]}$ *est* Lipschitz *sur un compact* $\mathcal{X} \subset \mathbf{R}^n$ *si pour tout* $[\mathsf{x}] \subset \mathcal{X}$, *il existe une constante* L *telle que* $w\left(\mathsf{f}_{[]}([\mathsf{x}])\right) \leq L w([\mathsf{x}])$. ♢

Remarque 2.6.1 *Dans le cadre des algorithmes sans branchements (instructions du type* `if...then...else`*) utilisant uniquement les opérations arithmétiques et les fonctions unaires usuelles* ($\sqrt{}, \sin, \exp, \ldots$), *Moore a établi dans [Moore79] les propriétés de convergence des fonctions d'inclusion naturelles et des formes centrées. Dans le cas d'une fonction d'inclusion naturelle*

$$d\left(f([x]), f_{[]}([x])\right) = O\left(w([x])\right), \tag{2.25}$$

et dans le cas d'une forme centrée

$$d\left(f([x]), f_{[],m}([x])\right) = O\left(w([x])^2\right). \tag{2.26}$$

Pour des intervalles $[x]$ *de faible longueur, la forme centrée peut donc être beaucoup moins pessimiste que*

la fonction d'inclusion naturelle. ♢

Pour améliorer l'encadrement fourni par une fonction d'inclusion sur un pavé, l'idée naturelle qui découle de cette remarque est de couper ce pavé en de petits pavés, voir l'exemple suivant

Exemple 2.6.2 *Soit la fonction* $f(x) = x^3 + x^2 - x + 1$. *Considérons la fonction d'inclusion naturelle* $f_{[]}([x])$ *de* $f(x)$ *et l'intervalle* $[x] = [-1, 1]$.

$$\begin{aligned} f([-1,1]) &= [-1,1]^3 + [-1,1]^2 - [-1,1] + 1 \\ &= [-1,1] + [0,1] - [-1,1] + 1 \\ &= [-1,4]\,. \end{aligned}$$

Coupons $[x]$ *en deux sous-intervalles* $[x]_1 = [-1, 0]$ *et* $[x]_2 = [0, 1]$ *et effectuons le même calcul sur ces deux sous-intervalles*

$$\begin{aligned} f_{[]}([x]_1) &= [0,3]\,, \\ f_{[]}([x]_2) &= [0,3]\,. \end{aligned}$$

Par conséquent, $f_{[]}([x]_1) \cup f_{[]}([x]_2) \subset f_{[]}([x])$. ♢

Pour un intervalle, on définit l'opération de *bissection* de la façon suivante

Définition 2.6.3 *Soit un intervalle* $[x]$, *non dégénéré. Bissecter* $[x]$ *consiste à le couper en deux sous-intervalles de même longueur* $[\underline{x}, m([x])]$ *et* $[m([x]), \overline{x}]$. ♢

Ainsi, lorsque l'intervalle $[-2, 4]$ est bissecté, les deux intervalles $[-2, 1]$ et $[1, 4]$ sont obtenus.

Or il existe de nombreuses manières de couper un pavé en deux sous-pavés, par conséquent, il convient de définir des règles systématiques pour réaliser cette opération : il faut choisir la composante qui sera bissectée. Plusieurs stratégies sont possibles, on peut bissecter le côté d de plus grande

- longueur : $d = d_\infty = \min\{i \,|w([x]_i) = w_\infty([\mathsf{x}])\ ,\ i = 1, \ldots, n\}$,
- longueur relative : $d = d_{\mathrm{rel}\infty} = \min\{i \,|w_{\mathrm{rel}}([x]_i) = w_{\mathrm{rel}\infty}([\mathsf{x}])\ ,\ i = 1, \ldots, n\}$,
- longueur pondérée : $d = d_{\mathrm{p}\infty} = \min\{i \,|w([x]_i \times p_i) = w_{\mathrm{p}\infty}([\mathsf{x}], \mathsf{p})\ ,\ i = 1, \ldots, n\}$.

Exemple 2.6.3 *Soit le pavé* $[\mathsf{x}] = \binom{[0.25,0.75]}{[1,2]}$. *Bissecter* $[\mathsf{x}]$ *suivant sa plus grande longueur fournit les deux pavés*

$$[\mathsf{x}]_1 = \binom{[0.25,0.75]}{[1,1.5]} \text{ et } [\mathsf{x}]_2 = \binom{[0.25,0.75]}{[1.5,2]}\,;$$

par contre, les pavés

$$[\mathsf{x}]_1 = \binom{[0.25,0.5]}{[1,2]} \text{ et } [\mathsf{x}]_2 = \binom{[0.5,0.75]}{[1,2]}$$

résultent d'une bissection de $[\mathsf{x}]$ *suivant son plus grande longueur relative.* ♢

Le choix de la direction de bissection, s'il n'a pas d'influence significative sur les résultats fournis par les algorithmes qui seront vus dans cette thèse, est dans la plupart des cas d'une certaine importance sur la vitesse d'exécution. Les méthodes de bissection seront comparées sur un exemple au chapitre 3.

2.7 Intervalles et tests booléens

Les fonctions d'inclusion vues jusqu'à maintenant faisaient intervenir des fonctions de variables réelles à valeur réelle. Or un certain nombre d'algorithmes ou de fonctions ont pour résultat des valeurs booléennes pour des entrées réelles ou booléennes. Pour utiliser les méthodes de l'analyse par intervalles dans ce contexte, il est commode d'introduire la notion d'*intervalle booléen* afin de tenir compte du fait que pour un vecteur intervalle donné, un test pourra être satisfait, non satisfait ou être indéterminé. La notion de test d'inclusion pourra ensuite être définie.

2.7.1 Intervalles booléens et tests d'inclusion

On considère l'ensemble des booléens $\mathbf{B} = \{0, 1\}$, avec 0 signifiant *faux*, et 1 *vrai*. Un *test booléen* est une fonction $t : \mathcal{D} \to \mathbf{B}$, avec $\mathcal{D}$ pouvant être inclus dans $\mathbf{R}^n, \mathbf{B}^n$ ou $\mathbf{R}^n \times \mathbf{B}^m$. Un intervalle booléen est un élément de $\mathbf{IB} = \{[0,0], [0,1], [1,1]\}$, où $[0,0]$ (qui sera noté 0) signifie toujours *faux*, $[1,1]$ (qui sera noté 1) signifie *vrai* et $[0,1]$ *indéterminé* (logique tri-valuée). Le tableau 2.1 présente le comportement des opérateurs ET ($\wedge$) et OU ($\vee$) appliqué à deux intervalles booléens

$\wedge$	1	0	$[0,1]$
1	1	0	$[0,1]$
0	0	0	0
$[0,1]$	$[0,1]$	0	$[0,1]$

$\vee$	1	0	$[0,1]$
1	1	1	1
0	1	0	$[0,1]$
$[0,1]$	1	$[0,1]$	$[0,1]$

Tableau 2.1: Opérations entre intervalles booléens.

Comme les intervalles booléens sont des ensembles, les opérations d'union et d'intersection peuvent leur être appliqués. Leur comportement ne doit pas être confondu avec celui des opérateurs logiques $\vee$ et $\wedge$. Par exemple, $[0,1] \wedge 1 = [0,1]$, par contre $[0,1] \cap 1 = 1$.

L'introduction des intervalles booléens permet de construire des fonctions d'inclusion pour les tests booléens.

Définition 2.7.1 *Un* test d'inclusion booléen *(test d'inclusion) d'un* test booléen $t : \mathcal{D} \to \mathbf{B}$ *est une fonction* $t_{[]} : \mathbf{I}\mathcal{D} \to \mathbf{IB}$ *telle que pour tout* $[\mathsf{x}] \in \mathbf{I}\mathcal{D}$, $t([\mathsf{x}]) \subset t_{[]}([\mathsf{x}])$, *c'est-à-dire*

$$\begin{aligned} t_{[]}([\mathsf{x}]) = 1 \quad &\Rightarrow \quad \forall \mathsf{x} \in [\mathsf{x}], t(\mathsf{x}) = 1, \\ t_{[]}([\mathsf{x}]) = 0 \quad &\Rightarrow \quad \forall \mathsf{x} \in [\mathsf{x}], t(\mathsf{x}) = 0. \end{aligned} \tag{2.27}$$

♢

Cette définition du test d'inclusion booléen est directement déduite de celle des fonctions d'inclusion classiques.

Exemple 2.7.1 *Un test d'inclusion $t_{[]}$ du test booléen $t(\mathsf{x}) = 1 \Leftrightarrow \mathsf{x} \in \mathcal{Y}$, où $\mathcal{Y}$ est un ensemble donné, est donné par*

$$\begin{cases} t_{[]}([\mathsf{x}]) = 1 & si\ [\mathsf{x}] \subset \mathcal{Y}, \\ t_{[]}([\mathsf{x}]) = 0 & si\ [\mathsf{x}] \cap \mathcal{Y} = \emptyset, \\ t_{[]}([\mathsf{x}]) = [0,1] & dans\ les\ autres\ cas. \end{cases} \tag{2.28}$$

♢

Soient $t_{[]1}$ et $t_{[]2}$ deux tests d'inclusion associés au même test booléen t. $t_{[]1}$ sera dit *plus fin* que $t_{[]2}$ si pour tout $[\mathsf{x}]$, $t_{[]1}([\mathsf{x}]) \subset t_{[]2}([\mathsf{x}])$. L'intersection de deux tests d'inclusion est plus fine que chacun des deux tests pris isolément, sauf dans le cas particulier où ils fournissent des résultats identiques. La proposition suivante sera utile pour construire des tests d'inclusion plus fins.

Proposition 2 *Soit $t_{[]}$ un test d'inclusion de t et $u_{[]}$ un test d'inclusion de u, tel que lorsque $t(\mathsf{x})$ est vrai alors $u(\mathsf{x})$ l'est également, on a alors $t'_{[]} = \left([0,1] \wedge u_{[]}\right) \cap t_{[]}$ est un test d'inclusion pour t, plus fin que $t_{[]}$.* ♢

Preuve : Si $u_{[]}([\mathsf{x}]) \in \{[0,1], 1\}$ alors $[0,1] \wedge u_{[]}([\mathsf{x}]) = [0,1]$ et $t'_{[]}([\mathsf{x}]) = [0,1] \cap t_{[]}([\mathsf{x}]) = t_{[]}([\mathsf{x}])$. Si $u_{[]}([\mathsf{x}]) = 0$, alors (2.27) s'applique et $\forall \mathsf{x} \in [\mathsf{x}]$, $u(\mathsf{x}) = 0$. De ce fait $\forall \mathsf{x} \in [\mathsf{x}]$, $t(\mathsf{x}) = 0$ (s'il existait $\mathsf{x}_0 \in [\mathsf{x}]$ tel que $t(\mathsf{x}_0)$ soit égal à 1, alors $u(\mathsf{x}_0)$ serait égal à 1). Comme $\forall \mathsf{x} \in [\mathsf{x}]$, $t(\mathsf{x}) = 0$, $t_{[]}([\mathsf{x}])$ vaut 0 ou $[0,1]$. Ainsi $t'_{[]}([\mathsf{x}]) = ([0,1] \wedge 0) \cap t_{[]}([\mathsf{x}]) = 0 \cap t_{[]}([\mathsf{x}]) = 0$, par conséquent $t'_{[]}([\mathsf{x}]) \subset t_{[]}([\mathsf{x}])$. Ainsi, $t'_{[]}([\mathsf{x}])$ est un test d'inclusion pour t, plus fin que $t_{[]}([\mathsf{x}])$. ■

Considérons un test booléen t. Une manière d'obtenir un test d'inclusion pour t est de remplacer chaque opérateur logique par son équivalent intervalle et de remplacer chaque occurrence d'une variable réelle (ou booléenne) dans l'expression du test par la variable intervalle correspondante. On obtient alors, ce que nous appelerons par analogie avec le cas réel, un *test d'inclusion naturel* $t_{[]}$ de t.

2.7.2 Tests d'inclusion particuliers

Le test booléen suivant est introduit pour le traitement de données aberrantes. Il sera mis en application aux chapitres 3 et 5

Soient n et q deux entiers strictement positifs, avec $q \leq n$, et n booléens t_i, $i = 1, \ldots, n$. On définit le test *et-q-relaxé*

$$\bigoplus\nolimits_q(t_1, \ldots, t_n) = \bigoplus_{i=1}^{n} {}_q(t_i) \tag{2.29}$$

qui vaut vrai si et seulement si au moins $n-q$ booléens t_i sont vrais. Lorsque $q=0$, $\bigoplus_q$ est équivalent à l'opérateur $\wedge$. Lorsque $q=n-1$, $\bigoplus_q$ est équivalent à l'opérateur $\vee$. Pour évaluer $\bigoplus_q$, soit s la somme, *au sens usuel du terme*, des valeurs des t_i, $\bigoplus_{i=1}^{n}{}_q(t_i)$ est vrai si et seulement si $s \geq n-q$.

Les règles usuelles peuvent être appliquées pour obtenir un test d'inclusion naturel pour $\bigoplus_q$. Ainsi, par exemple, $\bigoplus_{q[]}(0,[0,1],0,1)$ est égal à 1 si $q=1$, à $[0,1]$ si $q=2$ et à 0 si $q=3$, en effet, $[s]=[1,2]$.

Exemple 2.7.2 *Une pièce est équipée pour la prévention contre les incendies de n capteurs d'*absence *de fumée t_i disposés au plafond d'une pièce. Ces capteurs renvoient t_i = vrai lorsqu'ils ne captent pas de fumée. Certains de ces capteurs peuvent subir des dysfonctionnements (panne, présence d'un fumeur, ...), pour cette raison une alarme incendie n'est déclenchée que lorsque q capteurs au moins ne signalent plus l'absence de fumée. Ainsi, alarme $= \bigoplus_{i=1}^{n}{}_q(t_i)$. Les booléens intervalles permettent aussi de tenir compte de capteurs en panne à condition qu'un capteur hors service renvoie* $[0,1]$. ♢

Certains algorithmes font intervenir des branchements, c'est-à-dire que le résultat qu'ils fournissent dépend de la valeur de vérité de tests. Obtenir une fonction d'inclusion du résultat de tels algorithmes est relativement délicat. Une première solution est l'utilisation d'une fonction inspirée de la fonction χ de Kearfott [Kearfott95] : si t est le résultat d'un test booléen et y et z sont deux réels, alors

$$\chi(t,y,z) = \begin{cases} y & \text{si } t=1, \\ z & \text{si } t=0. \end{cases} \tag{2.30}$$

L'équivalent intervalle de la fonction $\chi\,(t,y,z)$ est donné par

$$\chi_{[]}([t]\,,[y]\,,[z]) = \begin{cases} [y] & \text{si } [t]=1, \\ [z] & \text{si } [t]=0, \\ [y] \underline{\cup} [z] & \text{si } [t]=[0,1]\,. \end{cases} \tag{2.31}$$

Le résultat d'une évaluation avec la fonction $\chi_{[]}$ est ainsi toujours un intervalle. L'extension de la fonction χ aux vecteurs et matrices est obtenue de manière évidente.

Le principal inconvénient d'une telle fonction est le fait que l'intervalle image qu'elle fournit peut éventuellement ne pas converger vers l'évaluation intervalle de la fonction χ à cause de l'opération $\underline{\cup}$. Pour cette raison, nous avons introduit la notion de fonctions d'inclusion *multivaluées*, l'image d'une telle fonction est un *ensemble* d'un ou plusieurs pavés. Dans le cas unidimensionnel, une nouvelle fonction d'inclusion multivaluée $\chi_{\mathrm{mv}[]}$ pour la fonction χ est définie de la manière suivante

$$\chi_{\mathrm{mv}[]}([t]\,,[y]\,,[z]) = \begin{cases} \{[y]\} & \text{si } [t]=1, \\ \{[z]\} & \text{si } [t]=0, \\ \{[y]\,,[z]\} & \text{si } [t]=[0,1]\,. \end{cases} \tag{2.32}$$

L'utilisation d'une telle fonction nécessite des soins particuliers pour le programme qui l'utilisera, ce

dernier devra en particulier tester si la fonction a un ou plusieurs pavés images, et le cas échéant, traiter chacun de ceux-ci.

Exemple 2.7.3 *Soit la fonction*

$$f(x) = \begin{cases} x+2 & si\ x \geq 0, \\ x-2 & si\ x < 0. \end{cases}$$

On peut la réécrire à l'aide de la fonction χ de la manière suivante

$$f(x) = \chi(x \geq 0, x+2, x-2).$$

Si $[x] = [-1,1]$, $f([x]) = [-3,-2[\cup [2,3]$. La fonction d'inclusion $f_{[]}$faisant intervenir $\chi_{[]}$ donne l'intervalle $[-3,3]$ et la fonction d'inclusion multivaluée $f_{mv[]}$ employant $\chi_{m[]}$ fournit $\{[-3,-1],[1,3]\}$ (voir la figure 2-5). Cette dernière évaluation est donc moins pessimiste. ♢

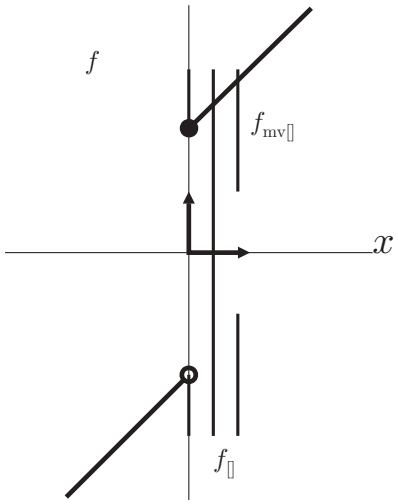


Figure 2-5: Exemple de l'utilisation d'une fonction d'inclusion multivaluée.

2.8 Mise en œuvre informatique

L'analyse par intervalle est restée relativement confidentielle dans un premier temps, sans doute à cause des difficultés de mise en œuvre de cette technique de calcul numérique. L'exemple suivant permettra de se rendre compte des problèmes d'implantation auxquels ont été confrontés les premiers analystes par intervalles.

Exemple 2.8.1 *Nous voulons construire une procédure qui calcule un prix en euros connaissant un prix en francs. L'algorithme de conversion franc-euro permet de réaliser cette tâche.*

Conversion franc-euro

```
entrées :      prix_en_francs;
               taux_de_change;
sortie :       prix_en_euros;
algorithme :   prix_en_euros = prix_en_francs * taux_de_change;
               renvoyer prix_en_euros;
```

Au 20 novembre 1998, le taux de change franc-euro n'est pas encore fixé, mais il sera situé dans un intervalle $[6.55\ F, 6.75\ F]$ *pour* 1 *euro. On pourra être intéressé par un intervalle dans lequel le prix en euros se trouvera certainement connaissant le prix en francs. L'algorithme suivant permettra d'effectuer cette opération.*

Conversion franc-euro, taux de change incertain (version 1970)

```
entrées :      prix_en_francs;
               taux_de_change_mini, taux_de_change_maxi;
sortie :       prix_en_euros_mini, prix_en_euros_maxi;
algorithme :   prix_en_euros_mini = prix_en_francs * taux_de_change_mini;
               prix_en_euros_maxi = prix_en_francs * taux_de_change_maxi;
               renvoyer prix_en_euros_mini, prix_en_euros_maxi;
```

Toutes les grandeurs manipulées étant positives, le prix en euros minimum sera le produit du prix en franc par le taux de change minimum, de même pour le prix en euros maximum. L'intervalle contenant le prix en euros [prix_en_euros_mini,prix_en_euros_maxi] *est ainsi obtenu.* ♢

Nous constatons que pour obtenir un encadrement du prix en euros, une récriture d'une partie de l'algorithme est nécessaire. Si cet exercice est envisageable pour des programmes de faible taille, dans le cas d'un programme complexe, il se révèle rapidement rédhibitoire. Dans le début des années 90, l'apparition de langages autorisant la surcharge d'opérateurs et la programmation orientée-objet a permis de créer un type intervalle, au même titre que le type entier ou flottant et de redéfinir par exemple l'addition, la soustraction, les fonctions usuelles pour des intervalles. Ainsi, le programme évaluant les bornes entre lesquelles se trouvera le prix en euros aura l'allure suivante

Exemple 2.8.2 (suite du 2.8.1) *Le taux de change appartient à* interval_taux_de_change.

Conversion franc-euro, taux de change incertain, (version intervalle 1990)

```
entrées :      prix_en_francs;
               interval_taux_de_change;
sortie :       interval_prix_en_euros;
algorithme :   interval_prix_en_euros = prix_en_francs * interval_taux_de_change;
               renvoyer interval_prix_en_euros;
```

L'algorithme obtenu ressemble donc beaucoup au premier algorithme de l'exemple 2.8.1 : les variables réelles ont été simplement remplacées par des variables intervalles. ♢

Dans une première partie de ce sous-chapitre, nous examinons la manière dont les ordinateurs représentent les réels et les difficultés qui découlent de cette représentation qui n'est qu'approchée. Consulter l'ouvrage de Pichat et Vignes [Pichat et al.93], ou les normes IEEE 754 [Ansi85] et 854 [Ansi87], ou encore l'ouvrage de Hammer *et al.* [Hammer et al.95] pour plus de détails.

Dans une seconde partie, nous examinerons comment, en C++ la notion de *classes* permet d'introduire les types intervalle, vecteur d'intervalles.

2.8.1 Représentation des réels en virgule flottante

Un nombre en virgule flottante est représenté sous la forme

$$x = \varepsilon m \times b^e = \varepsilon(0.m_1 m_2 ... m_i ...) \times b^e. \tag{2.33}$$

ε représente le *signe* du nombre, m sa *mantisse*, b la *base* et e désigne l'*exposant signé*. Lorsque $\frac{1}{b} \leq m < 1$, le flottant est dit *normalisé*. Cette normalisation garantit une représentation unique d'un réel par un flottant.

Les flottants sont généralement manipulés par les ordinateurs au format binaire ($b = 2$), avec une mantisse finie de longueur l. Ainsi, tout nombre qui aura une représentation en virgule flottante faisant intervenir plus de l chiffres significatifs (en base $b = 2$) ne pourra pas être représenté de manière exacte.

Exemple 2.8.3 *0.1 n'est pas représentable de manière exacte sur un ordinateur, dans un système binaire en virgule flottante. En effet, on peut décomposer ce nombre sous forme de puissances de 2 de la manière suivante*

$$0.1 = \sum_{k=1}^{\infty} (2^{-4k} + 2^{-(4k+1)}),$$

et ce développement est infini. ♢

L'exposant est également limité, $e_{\min} \leq e \leq e_{\max}$. Les notions d'infini et de limites n'ont alors plus de sens.

Exemple 2.8.4 *Les flottants en double précision conformes à la norme IEEE ont un exposant e compris entre -1021 et $+1024$, une mantisse limitée à $l = 53$; ils ne peuvent représenter que des nombres positifs situés dans l'intervalle* $\left[2.22 \times 10^{-308}, 1.80 \times 10^{+308}\right]$. ♢

2.8.2 Fonctions d'arrondi

En utilisant la représentation en virgule flottante avec précision limitée des réels, on approxime $\mathbf{R}$ par un sous-ensemble discret $\mathcal{R}$ (dit des réels représentables en machine). Les différentes opérations arithmétiques se faisant sur les flottants en précision finie, il faudra, d'une part, trouver une fonction permettant d'envoyer $\mathbf{R}$ sur $\mathcal{R}$ en perdant le moins d'information possible, et d'autre part, s'assurer que le résultat de l'évaluation des différentes fonctions mathématiques est obtenu avec le maximum de précision par rapport au calcul en précision illimitée. Des fonctions d'*arrondi* sont introduites à cet effet.

Définition 2.8.1 *Les fonctions* $\bigcirc$ *qui permettent d'envoyer* $\mathbf{R}$ *sur* $\mathcal{R}$ *sont appelées fonction d'arrondi si elles vérifient les propriétés suivantes :*

$$x \leq y \Rightarrow \bigcirc x \leq \bigcirc y, \tag{2.34}$$

$$\bigcirc x = x \Leftrightarrow x \in \mathcal{R}. \tag{2.35}$$

La fonction d'arrondi doit donc être *monotone* et laisser l'ensemble $\mathcal{R}$ invariant. La norme IEEE 754 définit différents types d'arrondis

Nom IEEE	alias	représentation
vers $-\infty$	vers le bas	$\triangledown$
vers $+\infty$	vers le haut	$\triangle$
optimal	au plus près	$\square$

L'arrondi au plus près vérifie la propriété d'*antisymétrie*

$$\square\,(-x) = -\,(\square\, x)\,. \tag{2.36}$$

Les arrondis orientés ($\triangledown$ et $\triangle$), par contre, ne satisfont pas cette propriété. Cependant, ils sont tels que

$$\begin{array}{l} \triangledown x \leq x \\ \triangle x \geq x \end{array} \text{ pour tout } x \in \mathbf{R}.$$

La figure ci-dessous illustre ces différentes formes d'arrondi (les barres verticales en gras signalent les éléments de $\mathcal{R}$, placés sur la droite réelle).
Deux éléments consécutifs de $\mathcal{R}$ sont séparés de 1 *ulp* (1 *unit at last place*). Remarquons que suivant la valeur de x, le diamètre de l'intervalle de 1 ulp défini par $[\triangledown x, \triangle x]$ est variable puisqu'il dépend de l'exposant.

Il faut redéfinir une arithmétique dans $\mathcal{R}$. Soit $\}$ l'opération en virgule flottante et précision finie associé à l'opération réelle $\circ \in \{+, -, \times, /\}$ et $\bigcirc$ l'arrondi correspondant ($\bigcirc \in \{\triangledown, \triangle, \square\}$) ; afin que

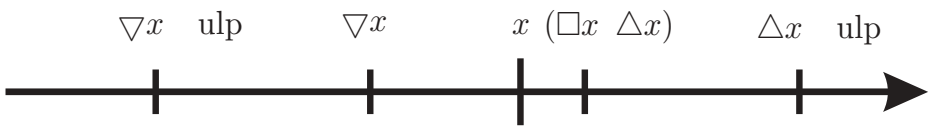


Figure 2-6: Différents types d'arrondis.

l'évaluation en flottant et précision finie se fasse avec le maximum de précision, il faut que la propriété suivante soit vérifiée

$$x \rbrace y = \bigcirc (x \circ y) . \tag{2.37}$$

Cela signifie que l'évaluation en flottant d'une fonction arithmétique doit correspondre à la valeur arrondie de la même opération réalisée en précision infinie.

Les fonctions vérifiant les propriétés (2.34), (2.35), (2.36) et (2.37) sont des *semimorphismes* [Hammer et al.95]. Toute fonction vérifiant ces propriétés donne un résultat avec la précision maximale, c'est-à-dire qu'il n'existe pas d'élément de $\mathcal{R}$ compris entre $x \rbrace y$ et $x \circ y$. Nous reprendrons la notation $\boxtimes (...)$ employée dans [Hammer et al.95], pour spécifier qu'une évaluation arithmétique s'est faite avec la précision maximale. De même, $\varphi_{\boxtimes}$ symbolisera l'évaluation en virgule flottante avec une précision maximale de la fonction d'une variable réelle φ, qui vérifie donc la propriété

$$\varphi_{\boxtimes} (x) = \boxtimes (\varphi(x)) .$$

2.8.3 Arithmétique sur les intervalles en virgule flottante

Comme précisé au chapitre précédent, nous serons amenés à utiliser des intervalles de réels qui seront manipulés par des ordinateurs. Le problème de la représentation mémoire de ces données se pose de la même manière que pour les réels.

Définissons l'ensemble des intervalles représentables en machine par

$$\mathbf{I}\mathcal{R} = \{[x] \in \mathbf{IR} | \underline{x} \in \mathcal{R}, \bar{x} \in \mathcal{R} \} . \tag{2.38}$$

Notons $\mathsf{3}$ l'arrondi pour les intervalles, qui vérifie les propriétés suivantes

$$[x] \subseteq [y] \Rightarrow \mathsf{3}\,[x] \subseteq \mathsf{3}\,[y] , \tag{2.39}$$

$$\mathsf{3}\,[x] = [x] \Leftrightarrow [x] \in \mathbf{I}\mathcal{R}. \tag{2.40}$$

Les opérations arithmétiques sur les intervalles flottants en précision limitée donneront des résultats avec

une précision maximale si

$$[x] \circ [y] = \mathsf{3}\,([x] \circ [y])\,. \tag{2.41}$$

De plus, l'évaluation fonctionnelle intervalle sur les flottants en précision finie d'une fonction φ d'une variable réelle conduira à un résultat de précision maximale lorsque

$$\varphi_{\mathsf{3}}([x]) = \mathsf{3}\,(\varphi([x]))\,.$$

Une évaluation avec une précision maximale signifie que le diamètre de l'intervalle solution est de 1 *ulp*.

2.8.4 Contrôle de la direction d'arrondi

Les processeurs mathématiques conformes aux normes IEEE 754 et 854 permettent de contrôler le type d'arrondi réalisé lors d'une opération mathématique. Cette possibilité sera mise en œuvre lorsqu'il faudra mettre en œuvre les opérations de base sur les intervalles. En effet, si l'on désire réaliser un algorithme de calcul garanti, il convient de maîtriser la gestion des arrondis afin d'obtenir des évaluations fiables.

La partie informatique de cette thèse s'est effectuée sur un PC disposant d'un processeur Pentium incluant un coprocesseur mathématique 80387. Ce coprocesseur dispose d'un mot de contrôle qui permet d'inhiber certaines interruptions (lors d'une division par zéro, d'un dépassement de capacité, etc.), de spécifier le nombre de bits des registres intermédiaires de calcul, et bien sûr de contrôler la direction des arrondis, conformément aux normes IEEE. Pour plus de détails sur le mot de contrôle et la programmation des 80x87, consulter l'ouvrage de Brumm et Brumm [Brumm et al.89], ou consulter le fichier d'en-tête <`float.h`> contenu dans la bibliothèque de tout compilateur C++.

En C++, le positionnement du contrôle d'arrondi se fera par une simple instruction, après avoir inclus le fichier d'en-tête <`float.h`> dans le programme utilisant le contrôle d'arrondi

```
_Control87(RC_UP,MCW_RC);                    //arrondi vers le haut
_Control87(RC_DOWN,MCW_RC);                   //arrondi vers le bas
_Control87(RC_NEAR,MCW_RC);                  //arrondi au plus près
```

Le contrôle d'arrondis sera illustré au paragraphe 2.8.5 par l'implantation de l'addition de deux intervalles.

La suite de ce chapitre est en partie inspirée des deux ouvrages de référence de la bibliothèque mathématique C-XSC, [Klatte et al.93] et [Hammer et al.95]. Nous allons présenter l'implantation de deux classes, la classe `interval`, pour la gestion des intervalles et la classe `ivector`, pour la mise en place des pavés. Cette présentation sera volontairement simplifiée, la description exhaustive de ces classes

ne présentant pas d'intérêt particulier. Nous nous contenterons d'insister sur les points délicats tels que la gestion de la mémoire et l'utilisation de variables temporaires.

2.8.5 Implantation d'une classe `interval`

Cette première classe est relativement aisée à mettre en œuvre. Elle ne nécessite aucune allocation dynamique de mémoire, les intervalles étant des données de taille fixe. Dans la partie déclarative, toutes les informations concernant la classe doivent être présentées. La déclaration d'une classe `interval` est illustrée dans le tableau 2.2.

```
class interval{
     double inf,sup;

public:
//constructeurs
     interval() //standard
          {inf=0;  sup=0; };
     interval(const double Binf, const double Bsup) //initialisé
          {inf=Binf;  sup=Bsup; };
     interval(const interval& Int) //par recopie
          {inf=Int.inf;  sup=Int.sup; };

//destructeur
     ~interval()
          {};

//fonctions membres
     friend interval operator+(const interval&,const interval&);
...
};
```

Tableau 2.2: Définition de la classe `interval`.

Les membres privés de la classe `interval` sont `inf`, et `sup`, les bornes inférieure et supérieure de l'intervalle (il aurait bien sûr également été possible de prendre le centre et le diamètre de l'intervalle). Trois constructeurs ont été présentés. Le constructeur standard initialise l'intervalle à 0 ; le constructeur initialisé donne les valeurs `Binf` et `Bsup` aux bornes de l'intervalle, enfin le constructeur par recopie crée une copie de l'intervalle `Int`. Le destructeur est vide, car il n'y a pas d'allocation dynamique de mémoire lors de l'appel de l'un des constructeurs de la classe `interval`. L'opérateur `+` a été déclaré ami de la classe, afin que les deux opérandes aient un rôle symétrique dans l'opération. Une définition en tant que fonction membre aurait également été possible.

L'addition est surchargée pour les intervalles par la fonction décrite dans le tableau 2.3.
Le passage des paramètres par référence (`interval&`) permet de ne pas recourir au constructeur par recopie lors de l'appel de la fonction. L'arrondi est orienté vers le bas avant l'ajoût des bornes inférieures et vers le haut avant l'ajoût des bornes supérieures. La syntaxe est sans surprise, voir la définition donnée dans (2.7). La structure pour une autre opération serait identique.

```
interval operator+(const interval& a,const interval& b)
{
     interval res;

     _Control87(RC_DOWN,MCW_RC);  //arrondi vers le bas
     res.inf = a.inf + b.inf;

     _Control87(RC_UP,MCW_RC);   //arrondi vers le haut
     res.sup = a.sup + b.sup;

     return (res);
}
```

Tableau 2.3: Surcharge de l'addition.

2.8.6 Implantation d'une classe `ivector`

Les vecteurs d'intervalles nécessitent un soin un peu plus particulier. En effet, la taille variable que peut prendre un vecteur nécessite une réservation dynamique de la mémoire qui lui sera allouée. Le tableau 2.4 illustre l'implantation de la classe `ivector`.

Les membres privés de la classe `ivector` sont deux entiers `lb` et `ub` indiquant les premier et dernier indices du vecteur, un pointeur `pdata` sur la première case-mémoire réservée aux données stockées dans le vecteur, et un drapeau `temp` qui prend la valeur `true` si le vecteur est ou non temporaire. Trois constructeurs sont présentés. Le constructeur standard crée un vecteur vide. Le constructeur à partir d'un entier réserve un tableau de `n` intervalles. Le constructeur par recopie est un peu plus délicat ; il réserve un tableau de même taille que celui du vecteur `a` puis il réalise une copie des éléments de `a`. Le destructeur libère l'espace mémoire éventuellement réservé (si `pdata` n'est pas le pointeur vide `NULL`). L'opérateur d'accès aux éléments du vecteur renvoie une référence sur l'intervalle se trouvant à la position `i` ; cette position correspond à la case mémoire réservée pointée par `pdata+i-lb`, car les indices commencent à `lb`. Une version sécurisée de l'opérateur d'accès aux éléments a été implantée en réalité, elle nécessite de tester si `i` est bien compris entre `lb` et `ub`. La compréhension de l'opérateur d'affectation, beaucoup plus délicat, exige l'examen préalable de la surcharge d'opérateurs arithmétiques.

Deux choix peuvent être considérés pour la surcharge des opérateurs et des fonctions renvoyant un vecteur. A titre d'exemple, nous allons les comparer sur le comportement de l'opérateur `+`. Une première syntaxe, décrite dans le tableau 2.5 utilise un retour par *valeur*.
La seconde syntaxe (tableau 2.6) fait intervenir un retour par *référence*.

Comme pour la version intervalle scalaire, le passage par référence des paramètres permet d'éviter un appel au constructeur par recopie lors de l'emploi de l'opérateur d'addition. Le résultat peut être retourné par valeur ou par référence.

Lorsque le résultat doit être retourné par valeur, une variable temporaire `res` est créée dans le corps

```
class ivector{
     interval* pdata;
     int lb,ub;
     bool temp;

public:
//constructeurs
     ivector() //standard
          {pdata=NULL;  lb=0;  ub=0;  temp=false; };
     ivector(int n) //initialisé
          {pdata=new interval[n];  lb=1;  ub=n;  temp=false};
     ivector(ivector& a) //par recopie
          { lb=a.lb;  ub=a.ub;  temp=false;
          data=new interval[ub-lb+1];
          for (int i=0; i<ub-lb; i++)
               *(pdata+i)=*(a.pdata+i);
          if (a.temp) delete a;
          };

//destructeur
     ~ivector()
          { if (pdata) delete[] pdata; };

//accès aux éléments du vecteur sans contrôle des bornes
     interval& operator[](int i)
          {return (*(pdata+i-lb)); };

//affectation
     void operator=(ivector&);  //détail hors déclaration

//fonctions membres et amies
     friend ivector& operator+(ivector&, ivector&);
     ...
};
```

Tableau 2.4: Définition de la classe `ivector`.

```
ivector operator+(ivector& a, ivector& b)
{
     ivector res(a.lb,a.ub);

     for (int i=lb;  i<=ub;  i++)
          res[i] = a[i] + b[i];

     return (res);
}
```

Tableau 2.5: Addition, retour d'une valeur.

```
ivector& operator+(ivector& a, ivector& b)
{
     ivector* pres = new ivector(a.lb,a.ub);
     pres->temp = true;

     for (int i=lb;  i<=ub;  i++)
          (*pres)[i] = a[i] + b[i];

     if (a.temp) delete a;
     if (b.temp) delete b;
     return (*pres);
}
```

Tableau 2.6: Addition, retour par référence.

de la fonction. Celle-ci sera détruite lors de la sortie de la fonction et une copie de `res` sera transmise au programme appelant.

Lorsque le résultat est retourné par référence, il faut réserver un espace mémoire où sont stockés les résultats de l'addition. La fonction renvoit une référence sur cet espace mémoire qui n'est pas libéré, à charge pour le programme appelant de le libérer. La seconde option est plus délicate à manipuler que la première, car la libération de la mémoire allouée au résultat de l'addition ne se fait pas dans la même fonction que sa réservation. En contrepartie, elle est plus rapide : si dans le premier cas deux instances du résultat sont créées, dans le second, une seule suffit. Le gain en vitesse d'exécution est de l'ordre de 30 %.

C'est pour indiquer au programme appelant que le résultat fourni est un vecteur temporaire qu'un drapeau `temp` a été ajoûté aux membres privés de `ivector`. `a` et `b` étant éventuellement des vecteurs temporaires résultats d'additions ou d'autres opérations, il faut tester leur drapeau et les détruire éventuellement, ce qui est fait juste avant la fin de la fin du corps de l'opérateur `+`. Généralement, c'est l'opérateur d'affectation (tableau 2.7) qui aura en charge la dernière destruction des variables temporaires.

```
void ivector:: operator=(ivector& a)
{
     if (this==&a) return;               // retour si affectation a=a

     if ((lb! =a.lb)||(ub! =a.ub))       // destruction-reallocation si
          { if (pdata) delete[] pdata;   // this n'a pas la taille de a
          lb=a.lb;  ub=a.ub;
          data=new interval[ub-lb+1]; }  // la taille est ajustée

     for (int i=0;  i<ub-lb;  i++)
          *(pdata+i)=*(a.pdata+i);

     if (a.temp) delete a;               // destruction de a si temporaire
}
```

Tableau 2.7: Opérateur d'affectation.

L'opérateur d'affectation ressemble beaucoup au constructeur par recopie. On commence par tester si l'affectation n'est pas du type `a=a;` il n'y aurait alors rien à faire. Si le vecteur destination n'a pas la même taille que le vecteur source, sa partie dynamique `pdata` est détruite et recréée à la bonne taille. Une copie élément par élément est effectuée, et le vecteur `a` est détruit s'il est temporaire. Cette structure n'est sans doute pas la plus efficace, mais elle représente un bon compromis entre simplicité et efficacité. Lorsque le vecteur `a` est temporaire, il est plus efficace de détruire l'espace pointé par `pdata` et de donner au pointeur la valeur de `a.pdata`, ainsi, en modifiant les cases mémoire pointées, on évite de copier chacun des éléments du tableau.

Insistons une nouvelle fois sur le danger que peut représenter l'usage de vecteurs temporaires définis comme précédemment. Soit la fonction `norme2`, amie de la classe `ivector`, qui calcule le carré de la norme d'un vecteur (tableau 2.8).

```
double norme2(ivector& a)
{
     double norme2=0;
     for (i=a.lb; i<=a.ub; i++)
          norme2=norme2+sqr(a[i]);
     return norme2;
}
```

Tableau 2.8: Calcul du carré de la norme d'un `ivector`.

Si une telle fonction est appelée par exemple par l'instruction

```
double a=norme(v+w);
```

la somme des deux vecteurs `v` et `w` est un vecteur temporaire qui n'est pas détruit dans la fonction `norme2`, par conséquent, l'espace mémoire alloué au résultat ne sera plus libéré. Si de nombreux appels de la fonction `norme` sont réalisés, la mémoire risque de saturer. Pour remédier à ce problème,on peut soit, tester le drapeau du vecteur `a`, en fin de fonction `norme2` et éventuellement le détruire s'il est temporaire

```
if (a.temp) delete a;
```

soit transmettre le vecteur `a` par valeur, l'en-tête de la fonction devient alors

```
double norme(ivector a);
```

la somme des deux vecteurs sera à nouveau stockée dans un vecteur temporaire. La fonction `norme` fera appel à un constructeur par recopie du vecteur temporaire qui le détruira, car la transmission se fait par valeurs. Le vecteur obtenu par recopie sera détruit à la fin de l'exécution de la fonction `norme2`.

2.8.7 Type intervalle booléen `boolinterval`

Nous avons vu au paragraphe 2.7, que les intervalles booléens permettent de manipuler facilement des fonctions de test sur des intervalles. La mise en œuvre informatique de ces objets est assez simple. Comme le type standard `bool` est une `enum` de deux valeurs `true` et `false`, le type `boolinterval` est une `enum` de trois valeurs `VRAI`, `FAUX` et `INDEF`. Ainsi,

```
enum boolinterval {VRAI, FAUX, INDEF};
```

permet de manipuler des intervalles booléens.

L'implantation d'un test d'inclusion booléen est alors très simple. Considérons le test du tableau 2.9 qui examine si un intervalle `a` est inclus dans un intervalle `b`. Nous supposons que cette fonction est déclarée amie (`friend`) de `interval`.

```
boolinterval inclus(interval& a, interval& b)
{
     if ((a.inf>=b.inf)&&(a.sup<=b.sup))
          return VRAI;
     if ((a.sup<b.inf)||(a.inf>b.sup))
          return FAUX;
     return INDEF;
}
```

Tableau 2.9: Test d'inclusion booléen.

Une version adaptée aux vecteurs intervalles serait très simple à obtenir à partir du test sur les intervalles, en appliquant ce dernier composante par composante.

2.9 Conclusion

Les bases de l'analyse par intervalles ont été rappelées et quelques notions nouvelles sur les fonctions d'inclusion d'algorithmes faisant intervenir les intervalles booléens et les tests ont été présentés.

Différents aspects ou variantes de l'analyse par intervalles n'ont délibérément pas été présentés. Par exemple, la *modal interval analysis* (voir [Gardenes et al.85] ou [Vehi98]) considère deux types d'intervalles, les intervalles *propres*, correspondant aux intervalles classiques et les intervalles *impropres*, dont l'interpétation est légèrement différente. Un intervalle propre représente un réel dont la valeur est incertaine, un intervalle impropre désigne tous les réels compris entre ses bornes. Un certain nombre de propriétés découlent de ces considérations qui n'ont pas été employés dans le cadre de ce document. Signalons également l'existence de l'*affine interval arithmetic* (voir par exemple [Andrade et al.94]), permettant de stocker la dépendance entre variables dans une expression et apportant une solution, au moins dans le cas linéaire, au problème du pessimisme introduit lors de la manipulation d'occurences multiples d'une variable. Ces aspects n'intervenant pas dans la suite de l'exposé, nous avons pris le parti de ne pas les détailler.

Le choix qui a été fait pour la gestion des vecteurs est sans doute discutable. Fallait-il employer une structure plus complexe que celle qui est rencontrée dans bon nombre d'ouvrages dédiés à la programmation en C++, pour obtenir un gain de vitesse sensible, mais peut-être pas indispensable? Pour les vecteurs, qui demeurent des structures relativement légères, le choix de la simplicité aurait pu être fait. Par contre, pour les structures d'arbres qui seront utilisées au chapitre 5, le choix qui a été fait pour les vecteurs s'impose.

Chapitre 3

Estimation paramétrique non-linéaire

3.1 Introduction

Qu'un biologiste veuille analyser le devenir d'un médicament dans un organisme, qu'un physicien veuille prévoir la quantité de radiations émises par un échantillon radioactif, ou qu'un électrotechnicien désire commander un moteur électrique, une étape de modélisation préalable de ces systèmes est nécessaire. La modélisation consiste à trouver des représentations mathématiques pour rendre compte du comportement du système que l'on aura éventuellement isolé de l'ensemble plus vaste, donc plus complexe, dont il fait partie.

En général, un modèle est adapté à une classe de systèmes : la structure globale d'un modèle physique restera par exemple identique pour une machine asynchrone de faible puissance ou de plus forte puissance. Seules les grandeurs physiques (résistances, inductances ...) intervenant dans la modélisation (ce sont les *paramètres* du modèle) vont varier. Il va falloir identifier ces paramètres afin d'obtenir une description satisfaisante du processus.

Exemple 3.1.1 *Considérons un échantillon de substance radioactive, l'évolution de la quantité d'isotope radioactif y est décrite par une équation différentielle du premier ordre*

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -ky, \tag{3.1}$$

on en déduit

$$y(t) = y_0 \exp(-kt), \ \textit{pour } t \geq 0. \tag{3.2}$$

Cette modélisation résulte de l'observation, elle est essentiellement descriptive, c'est un modèle comporte-

mental. *Le paramètre k, qui dépend de la substance considérée, intervient de manière non-linéaire dans ce modèle, contrairement au paramètre y_0. Le modèle est donc partiellement non-linéaire en les paramètres.*$\diamondsuit$

Des expériences, des essais sont réalisés sur le système réel afin d'observer le comportement de ses *sorties*, c'est-à-dire de grandeurs physiques accessibles à la mesure (température, pression, vitesse, etc.), en réponse à des *entrées* qui correspondent à des grandeurs du système que l'on connaît et que l'on peut maîtriser. Les mesures réalisées vont permettre d'identifier la valeur des différents paramètres. Par la suite, les sorties mesurées sur le système seront notées $\mathsf{y}_\mathrm{s}(t_i)$, les t_i représentant les paramètres de l'expérience (instants de mesures, positions du capteur, fréquences considérées ...) ; les sorties prévues par le modèle sont $\mathsf{y}_\mathrm{m}(t_i, \mathsf{p})$, où p est le vecteur des paramètres. Enfin la différence entre les sorties mesurées et les sorties prédites par le modèle sera notée $\mathsf{e}(t_i, \mathsf{p})$ avec

$$\mathsf{e}(t_i, \mathsf{p}) = \mathsf{y}_\mathrm{s}(t_i) - \mathsf{y}_\mathrm{m}(t_i, \mathsf{p}). \tag{3.3}$$

Cette erreur correspond d'une part à notre aptitude limitée à modéliser les phénomènes physiques les plus fins intervenant dans le processus à décrire (on parle d'*erreur structurelle*) et d'autre part aux *erreurs de mesure*, c'est-à-dire à un bruit venant entacher les mesures prélevées (précision limitée de l'appareil, perturbations, etc.). En regroupant toutes les erreurs correspondant aux n valeurs prises par t_i, on peut écrire à partir de (3.3)

$$\mathsf{e}(\mathsf{p}) = \mathsf{y}^\mathrm{s} - \mathsf{y}^\mathrm{m}(\mathsf{p}), \tag{3.4}$$

avec $\mathsf{e}(\mathsf{p}) = \left[\mathsf{e}^\mathrm{T}(t_1, \mathsf{p}), \ldots, \mathsf{e}^\mathrm{T}(t_n, \mathsf{p})\right]^\mathrm{T}$, $\mathsf{y}^\mathrm{s} = \left[\mathsf{y}_\mathrm{s}^\mathrm{T}(t_1), \ldots, \mathsf{y}_\mathrm{s}^\mathrm{T}(t_n)\right]^\mathrm{T}$ et $\mathsf{y}^\mathrm{m}(\mathsf{p}) = \left[\mathsf{y}_\mathrm{m}^\mathrm{T}(t_1, \mathsf{p}), \ldots, \mathsf{y}_\mathrm{m}^\mathrm{T}(t_n, \mathsf{p})\right]^\mathrm{T}$.

Deux types de structures de modèle peuvent être considérées. La structure de *modèle linéaire* en les paramètres est la plus simple, elle peut s'écrire

$$\mathsf{y}_\mathrm{m}(t_i, \mathsf{p}) = \mathsf{R}^\mathrm{T}(t_i).\mathsf{p}, \tag{3.5}$$

avec $\mathsf{R}(t_i)$ une matrice de régresseurs connue pour tout t_i. Les *modèles non-linéaires* en les paramètres ont une structure beaucoup plus libre, de la forme

$$\mathsf{y}_\mathrm{m}(t_i, \mathsf{p}) = \mathsf{f}(t_i, \mathsf{p}), \tag{3.6}$$

où $\mathsf{f}(t_i, .)$ est une fonction quelconque, dont la forme est également connue pour tout t_i.

Parmi les nombreuses méthodes d'identification disponibles (voir [Walter et al.97] par exemple), beaucoup passent par la minimisation d'une fonction coût faisant intervenir le vecteur des paramètres. Tout argument $\hat{\mathsf{p}}$ de l'optimum de ce critère correspond à une valeur *optimale* des paramètres, mais uniquement *au sens du critère* retenu, en effet, la valeur de $\hat{\mathsf{p}}$ peut varier fortement suivant la fonction coût choisie.

D'autres méthodes recherchent les valeurs des paramètres *compatibles*, en un sens à définir, avec les

valeurs des erreurs de sortie ou de mesure, supposées appartenir à des intervalles connus ; on parle d'estimation à *erreur bornée*. L'estimée des paramètres ne sera alors plus ponctuelle, mais constituée d'un ensemble de valeurs.

Le choix de la méthode à employer pour la recherche des paramètres sera en partie dicté par les hypothèses faites sur l'erreur de sortie, mais également par l'usage qui sera fait de l'estimée obtenue. Par exemple, lorsqu'une valeur ponctuelle suffit, les méthodes par minimisation seront envisageables. Lorsqu'il faudra garantir une propriété pour toutes les valeurs possibles des paramètres, disposer de l'ensemble des valeurs compatibles avec les mesures s'avère très utile.

Dans une première partie de ce chapitre, nous allons présenter une technique d'identification par minimisation de critère. Le paragraphe 3.2 sera consacré à la présentation d'un algorithme classique d'optimisation globale garantie utilisant les outils de l'analyse par intervalles (voir également [Mckinnon et al.96] et [Kieffer et al.98c]). Cet outil sera appliqué à l'identification des paramètres de modèles compartimentaux, relativement répandus en biologie et en pharmacocinétique, au paragraphe 3.3. Dans une seconde partie, le paragraphe 3.4 traitera d'un algorithme d'inversion ensembliste adapté au contexte de l'identification à erreurs bornées. Enfin, au paragraphe 3.5, l'identification des paramètres d'un modèle comportemental sera abordée dans le contexte erreurs bornées.

Dans la suite de ce chapitre, les sorties $\mathsf{y}_{\mathrm{s}}(t_i)$ et $\mathsf{y}_{\mathrm{m}}(t_i, \mathsf{p})$ ainsi que l'erreur $\mathsf{e}(t_1, \mathsf{p})$ seront supposés scalaires, afin de simplifier la présentation.

3.2 Estimation par minimisation d'un critère

3.2.1 Fonction coût

La valeur des paramètres optimaux va dépendre assez fortement de la forme choisie pour la fonction coût $j(\mathsf{p})$. On cherchera le plus souvent à obtenir à chaque instant une sortie prédite par le modèle $\mathsf{y}^{\mathrm{m}}(\mathsf{p})$ aussi proche que possible des sorties mesurées y^{s}. Pour cela, on pourra par exemple former la somme des carrés des différences entre les sorties du modèle et les mesures

$$j_{\mathrm{mc}}(\mathsf{p}) = \mathsf{e}^{\mathrm{T}}(\mathsf{p})\,\mathsf{e}(\mathsf{p})\,; \tag{3.7}$$

on obtient alors un critère des *moindres carrés*. La forme plus générale de ce critère est

$$j_{\mathrm{mcp}}(\mathsf{p}) = \mathsf{e}^{\mathrm{T}}(\mathsf{p})\,\mathsf{Q}\,\mathsf{e}(\mathsf{p})\,, \tag{3.8}$$

où Q est une matrice symétrique définie non négative. La valeur des paramètres correspondant au minimum de ce critère sera dite optimale au sens des moindres carrés pondérés. Lorsque Q est diagonale, on

peut réécrire (3.8) sous la forme

$$j_{\mathrm{mcp}}(\mathsf{p}) = \frac{1}{n}\sum_{i=1}^{n} w_i e^2(t_i, \mathsf{p}). \tag{3.9}$$

Les *poids* w_i permettent de donner une plus ou moins grande importance à certaines erreurs. Des critères faisant intervenir la valeur absolue de l'erreur peuvent également être utilisés, qui pénalisent de manière moins importante les très grandes valeurs de l'erreur. On pourra par exemple écrire

$$j_{\mathrm{absp}}(\mathsf{p}) = \frac{1}{n}\sum_{i=1}^{n} w_i \left|e(t_i, \mathsf{p})\right|. \tag{3.10}$$

Cependant, ce type de critère présente l'inconvénient de ne pas être différentiable en tout point où l'une des erreurs s'annule, ce qui rend délicate l'obtention de l'optimum.

De très nombreuses techniques (méthodes de Newton, de Gauss-Newton, du recuit simulé, etc.) proposent de résoudre le problème de la recherche de l'optimum d'un critère. Qu'elles soient déterministes ou stochastiques, la plupart font intervenir des calculs dits *ponctuels*. Les fonctions à minimiser ne sont évaluées que sur un nombre fini de points. Sauf en des cas particuliers, il n'est pas possible d'en tirer des conclusions sur *toutes* les valeurs de la fonction. Dans le meilleur des cas, la valeur des paramètres obtenue sera optimale au sens du critère (par exemple des moindres carrés). Cependant, il peut arriver que cet optimum soit délicat à atteindre, en particulier lorsque la sortie du modèle n'est pas linéaire en les paramètres ; le critère présente alors parfois plusieurs minima locaux non globaux. Dans ces conditions, les méthodes de recherche d'optima de critères étant pour la plupart locales, elles ne garantissent en rien la validité du résultat obtenu, la valeur du critère considérée optimale pouvant en fait correspondre à l'un de ces minima locaux.

Dans le paragraphe suivant, nous allons présenter les grandes lignes de l'algorithme d'optimisation globale dû à Hansen, qui évite ces inconvénients. Cet algorithme, utilisant les outils de l'analyse par intervalles et en particulier la notion de fonction d'inclusion, propose en effet de fournir un encadrement de *tous* les arguments de l'optimum d'une fonction, contrairement aux méthodes classiques.

3.2.2 Algorithme d'optimisation globale

Seul le principe d'une version simplifiée de l'algorithme de Hansen sera présenté ici. On pourra consulter l'ouvrage de référence [Hansen92] pour plus de détails ; une présentation particulièrement pédagogique et complète de l'algorithme est faite dans [Didrit97] ; [Hammer et al.95] en donne une version simplifiée très proche de celle utilisée dans la suite de ce document.

L'idée fondatrice de l'algorithme est illustrée par l'exemple suivant.

Exemple 3.2.1 *Considérons la fonction* $f(x) = x^4 - 4x^2$ *représentée sur la figure 3-1. Supposons que l'on recherche le minimum de cette fonction sur* $[-2, 3]$.

A l'aide de la fonction d'inclusion naturelle $f_{[]}([x]) = [x]^4 - 4[x]^2$, *on prouve que* $f([2,3]) \subset f_{[]}([2,3]) =$

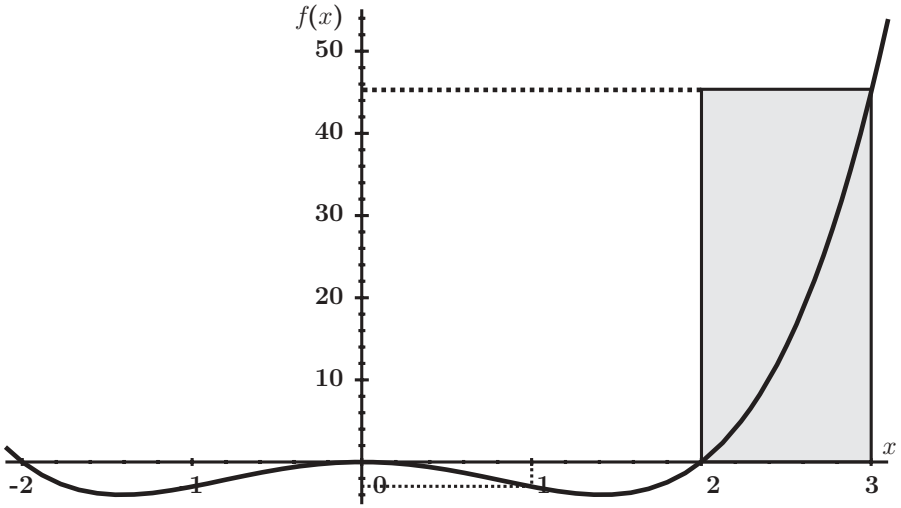


Figure 3-1: Représentation graphique de $f(x) = x^4 - 4x^2$.

$[0, 45]$ *; comme* $f(1) = -3$*, on* prouve *ainsi* numériquement *que le minimum de* $f(x)$ *sur* $[-2, 3]$ *ne se trouve pas dans l'intervalle* $[2, 3]$*. On poursuit ensuite le traitemet sur les intervalles* $[-2, -1]$, $[-1, 0]$, *etc.* ♢

Cette technique de démonstration numérique sera employée dans l'algorithme de Hansen. Soit une fonction deux fois différentiable $f : \mathbf{R}^n \to \mathbf{R}$. Considérons le problème de la recherche de l'argument de son minimum *global non contraint* à l'intérieur d'un pavé de recherche donné $[\mathsf{x}]_0 \in \mathbf{IR}^n$. L'objectif de l'algorithme de Hansen est d'encadrer tous les minimiseurs $\mathsf{x}^* \in]\mathsf{x}[_0$ tels que

$$f^* = f(\mathsf{x}^*) = \min_{\mathsf{x} \in]\mathsf{x}[_0} f(\mathsf{x}) \tag{3.11}$$

dans un ensemble de pavés. Le cas d'un minimiseur se trouvant sur la frontière de $[\mathsf{x}_0]$ n'est pas considéré ici. Le pavé $[\mathsf{x}]_0$ peut avoir une taille arbitrairement grande, il ne constitue de ce fait pas une contrainte sur la localisation du ou des arguments du minimum global.

Schématiquement, l'algorithme va couper $[\mathsf{x}]_0$ en sous-pavés $[\mathsf{x}]_i \subset [\mathsf{x}]_0$ et les stocker dans une liste $\mathcal{L}$ de pavés à traiter. Tout pavé pour lequel l'algorithme aura prouvé qu'il ne contient pas le minimiseur global sera éliminé de $\mathcal{L}$. Pour ce faire, trois tests d'élimination sont considérés qui seront détaillés plus loin : le test du point milieu, le test de monotonicité et le test de convexité. Ils utilisent respectivement une fonction d'inclusion de la fonction, de son gradient et d'éléments de son Hessien.

A chaque itération de l'algorithme, un pavé $[\mathsf{x}]$ est extrait de la liste $\mathcal{L}$. Une tentative d'amélioration de $\tilde{f}$, majorant courant du minimum global de f, est effectuée en appliquant une méthode de Newton standard sur $[\mathsf{x}]$. Ensuite, les trois tests d'élimination sont appliqués à $[\mathsf{x}]$.

Si $[\mathsf{x}]$ n'a pas été éliminé, la version *intervalle* de l'algorithme de Newton-Gauss-Seidel est utilisée. Cette étape a pour ambition d'une part de réduire la taille de $[\mathsf{x}]$ et d'autre part éventuellement de le

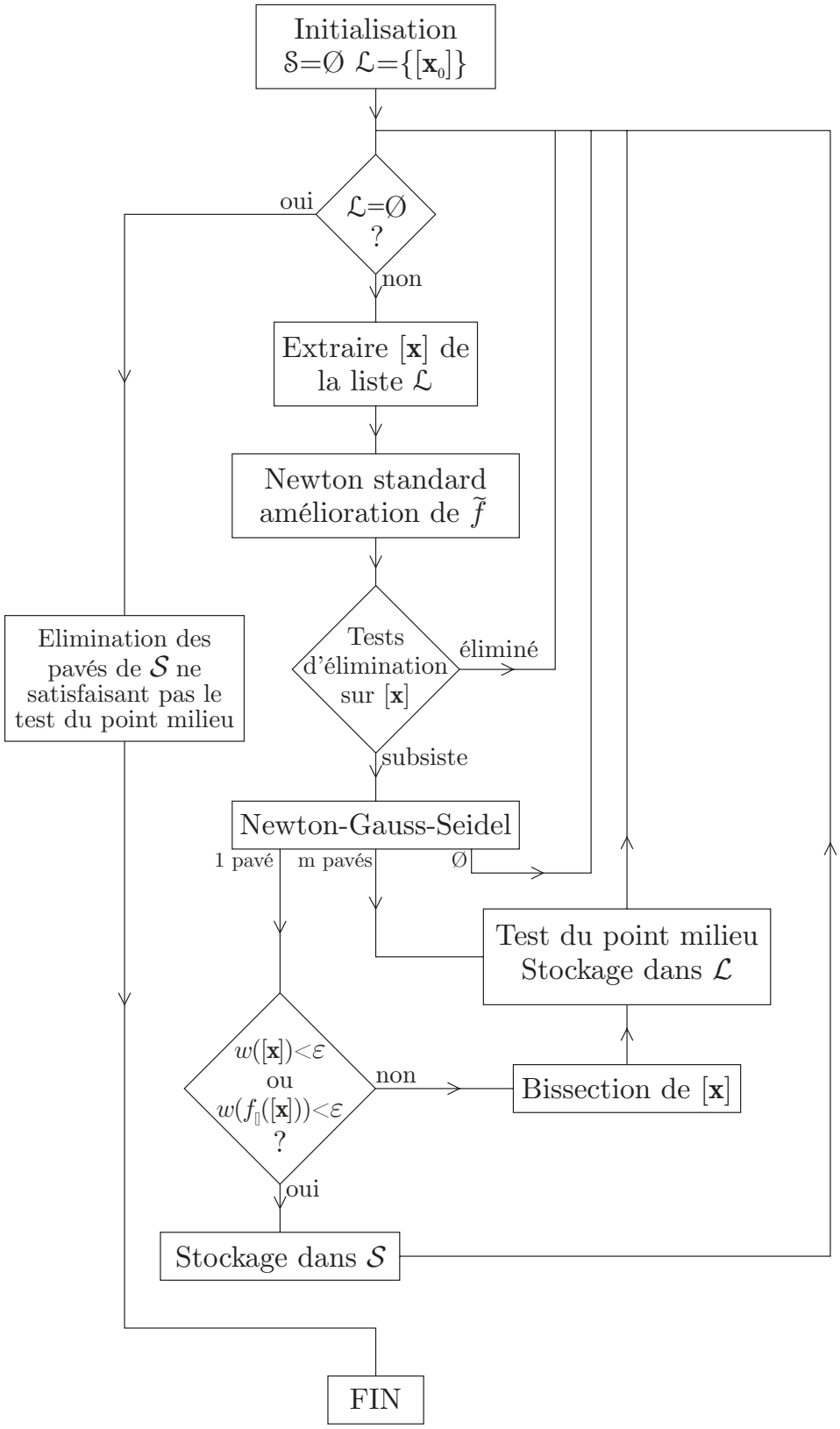


Figure 3-2: Schéma de l'algorithme de Hansen simplifié.

scinder en plusieurs sous-pavés lorsque plusieurs minimiseurs appartiennent à $[\mathsf{x}]$.

Lorsque plusieurs sous-pavés $[\mathsf{x}]'_i$ sont ainsi obtenus, leur est associée une borne inférieure du critère sur chacun d'eux, évaluée grâce à $f_{[]}$. Si un seul pavé réduit $[\mathsf{x}]'$ résulte de l'itération de Newton-Gauss-Seidel, et si sa longueur maximale est inférieure à ϵ, ou si la longueur de l'intervalle fourni par la fonction $f_{[]}$ sur $[\mathsf{x}]'$ est plus petite que ϵ, $[\mathsf{x}]'$ est placé dans la liste des solutions potentielles $\mathcal{S}$. Dans le cas contraire, il est coupé en deux sous-pavés $[\mathsf{x}]'_1$ et $[\mathsf{x}]'_2$. Dans les deux cas, les pavés $[\mathsf{x}]'_i$ résultants sont, dans la mesure où ils passent le test du point milieu, stockés dans la liste $\mathcal{L}$.

L'algorithme est ensuite itéré en considérant un nouveau pavé $[\mathsf{x}]$ extrait de la liste $\mathcal{L}$; il s'arrête lorsque $\mathcal{L}$ est vide. Il faut alors effectuer un test du point milieu sur tous les pavés stockés dans $\mathcal{S}$, car en fin d'algorithme, on dispose du plus petit majorant du minimum du critère et certains pavés ont pu être stockés dans $\mathcal{S}$ alors que cette borne supérieure n'était pas disponible. Finalement, $\mathcal{S}$ contient un ensemble de pavés encadrant les arguments du minimum global. L'algorithme de Hansen simplifié est illustré sur la figure 3-2, chacune de ses étapes sera détaillée dans les sous-paragraphes suivants, où le pavé courant à traiter sera noté $[\mathsf{x}]$.

Test du point milieu

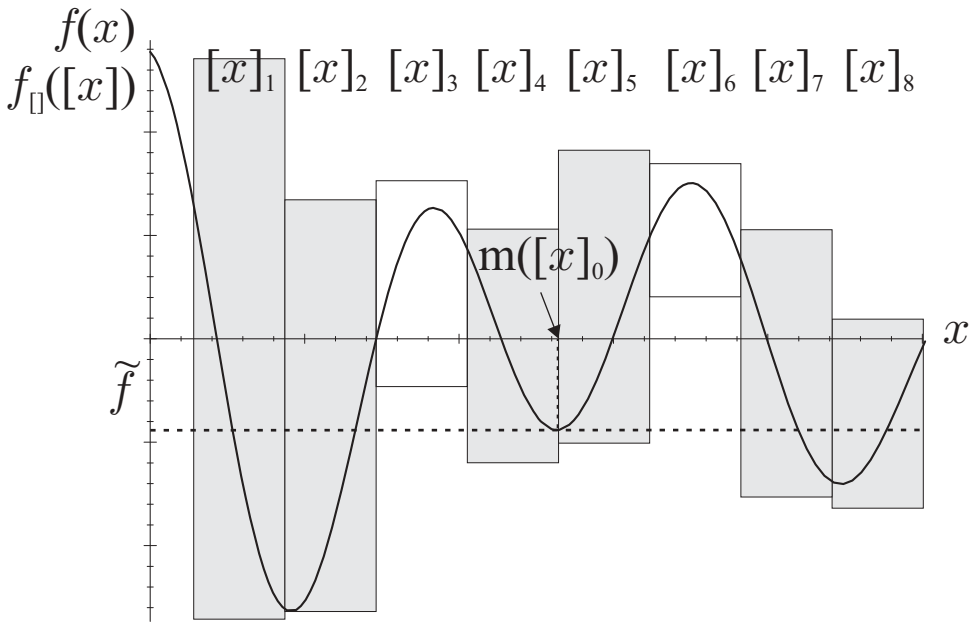


Figure 3-3: Test du point milieu. Les intervalles associés à des rectangles en blanc peuvent être éliminés.

Ce test suppose que l'on dispose d'un majorant $\tilde{f}$ du minimum f^* de f sur $[\mathsf{x}_0]$. Lors de l'initialisation, on pourra considérer par exemple $\tilde{f} = f(\mathsf{m}([\mathsf{x}_0]))$, où $\mathsf{m}([\mathsf{x}_0])$ désigne le milieu de $[\mathsf{x}_0]$ et donne son nom au test. Considérons un pavé courant $[\mathsf{x}]$ issu de la liste $\mathcal{L}$ et notons $f_{[]}([\mathsf{x}]) = [\underline{f_x}, \overline{f_x}]$, l'image de

$[\mathsf{x}]$ par la fonction d'inclusion de f. Si

$$\underline{f_x} > \tilde{f}, \tag{3.12}$$

alors $[\mathsf{x}]$ ne peut pas contenir de minimiseur global et doit de ce fait être éliminé de $\mathcal{L}$.

Supposons par exemple que l'on cherche à minimiser la fonction f dont le graphe est représenté sur la figure 3-3. $\tilde{f}$ est disponible et a été évalué au centre de $[x_0]$. Cet intervalle a été divisé en huit sous-intervalles. Une évaluation de l'image de ces sous-intervalles est disponible grâce à une fonction d'inclusion de f ; cette image correspond au côté vertical de chacun des rectangles. Les intervalles $[x]_3$ et $[x]_6$ satisfaisant (3.12) ne peuvent contenir aucun minimiseur global de f ; ils doivent donc être éliminés.

Test de monotonicité

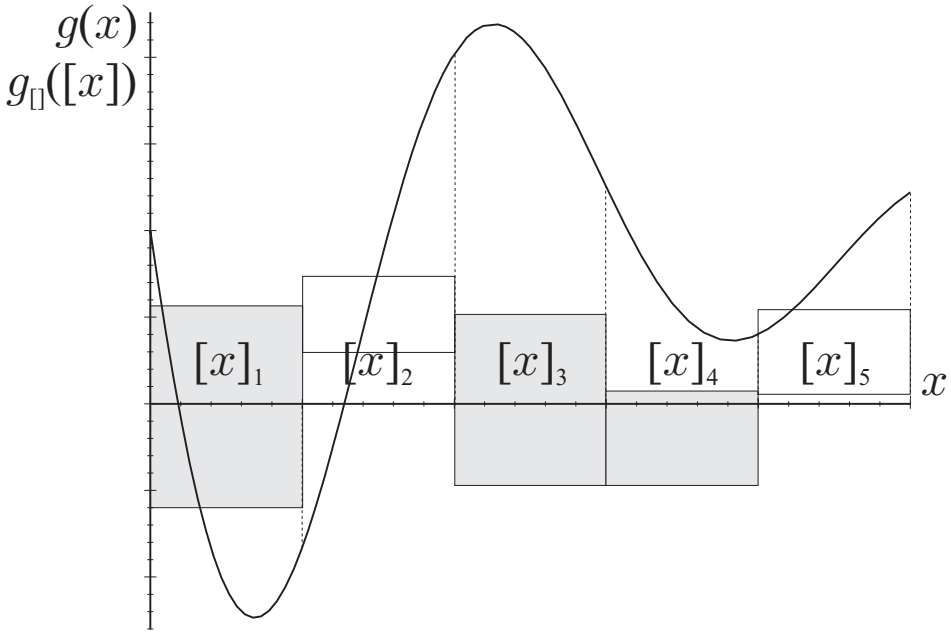


Figure 3-4: Test de monotonicité. Les intervalles associés à des rectangles en blanc peuvent être éliminés.

Lorsque la fonction d'inclusion du gradient de f est telle que

$$0 \notin \mathsf{g}_{[]}\left([\mathsf{x}]\right), \tag{3.13}$$

où $\mathsf{g}_{[]} = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)^{\mathrm{T}}_{[]}$, alors f est strictement monotone sur $[\mathsf{x}]$. De ce fait $[\mathsf{x}]$ ne peut contenir de point stationnaire et il peut être éliminé de $\mathcal{L}$, le cas de minimiseurs sur la frontière de $[\mathsf{x}]_0$ n'étant pas considéré.

La figure 3-4 présente par exemple la dérivée $g\left(x\right)$ d'une fonction unidimensionnelle $f\left(x\right)$ à minimiser. Les images de cinq intervalles par la fonction d'inclusion de $g\left(x\right)$ ont également été représentées. $[x]_2$ et

$[x]_5$ ne peuvent contenir de minimiseurs car leur dérivée ne contient pas 0 ; ils n'ont donc plus à être considérés lors des recherches ultérieures.

Test de convexité

Si x^* est un minimiseur non contraint de f, alors f doit être convexe dans un voisinage de x^*. Cela implique que sa matrice hessienne H, évaluée en x^*, est définie non négative. Une condition nécessaire en est que les éléments diagonaux de H, H_{ii} $(i = 1, ..., n)$ soient positifs ou nuls. Par conséquent, si la fonction d'inclusion de la matrice hessienne de f en $[\mathsf{x}]$ est telle que

$$\exists i \in \{1, ..., n\} \text{ tel que } \mathsf{H}_{ii[]}([\mathsf{x}]) \subset \mathbf{R}^*_-, \tag{3.14}$$

avec $\mathsf{H}_{ii[]} = \left(\frac{\partial^2 f}{\partial x_i^2}\right)_{[]}$, alors f ne peut pas être convexe sur $[\mathsf{x}]$, et $[\mathsf{x}]$ peut de ce fait être éliminé.

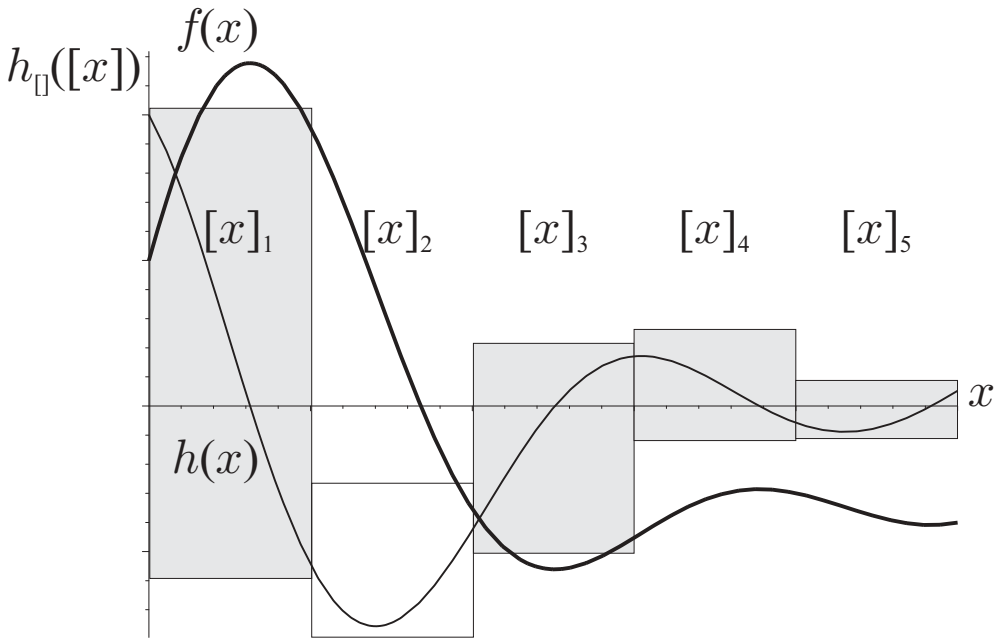


Figure 3-5: Test de convexité. L'intervalle associé au rectangle en blanc peut être éliminé.

La figure 3-5 présente le graphe d'une fonction f à minimiser ainsi que le graphe de sa dérivée seconde $h(x)$. L'image par une fonction d'inclusion de la dérivée seconde a été évaluée pour cinq intervalles. Pour l'intervalle $[x]_2$, la dérivée seconde de f est strictement négative, par conséquent la fonction n'est pas convexe sur cet intervalle qui ne peut donc pas contenir de minimiseur global.

Réduction des pavés

Cette étape va permettre d'une part, de réduire la taille des pavés de recherche, d'autre part, de séparer d'éventuels minimiseurs locaux. L'algorithme de Newton-Gauss-Seidel, dont seules les propriétés seront

présentées, est une généralisation au cas multidimentionnel de l'algorithme de Newton par intervalles (voir [Moore66], [Hansen92] ou [Hammer et al.95]) qui sera quant à lui détaillé.

Algorithme de Newton par intervalles Dans le cas unidimensionnel, l'argument du minimum global d'une fonction ne peut se trouver que dans un intervalle sur lequel sa dérivée s'annule (voir le test (3.13)). Les points pour lesquels cette dérivée s'annule constituent au contraire des candidats potentiels pour l'argument du minimum global. La recherche des zéros de la dérivée est accélérée par la méthode de Newton par intervalles.

Considérons une fonction scalaire $\varphi : \mathbf{R} \rightarrow \mathbf{R}$, continûment différentiable. La résolution de

$$\varphi(x) = 0 \tag{3.15}$$

sur l'intervalle $[x] \in \mathbf{IR}$ peut être réalisée à l'aide de la formule de la valeur moyenne, soit $x^* \in [x]$, il existe $\xi \in [x]$ tel que

$$\varphi(m([x])) - \varphi(x^*) = \varphi'(\xi) \cdot (m([x]) - x^*) . \tag{3.16}$$

En supposant que x^* est un zéro de φ, il est possible d'obtenir

$$x^* = m([x]) - \frac{\varphi(m([x]))}{\varphi'(\xi)} \in m([x]) - \frac{\varphi(m([x]))}{\varphi'_{[]}([x])} = N([x]) . \tag{3.17}$$

$N([x])$ est l'intervalle obtenu par une itération de l'algorithme de Newton. De ce fait, tout zéro de $\varphi(x)$ se trouvant dans $[x]$ se trouve également dans $N([x])$, et par conséquent dans $[x] \cap N([x])$. Il est ainsi possible itérativement d'obtenir un encadrement de plus en plus fin d'un zéro se trouvant dans $[x]$ en appliquant l'algorithme suivant

Algorithme 1 *(**Newton-Intervalle**) soit $[x]^0$ un intervalle de recherche d'un zéro de φ tel que $0 \notin \varphi'\left([x]^0\right)$, alors*

$$[x]^{k+1} = [x]^k \cap N\left([x]^k\right), \ k = 0, 1, \ldots \tag{3.18}$$

est une suite d'intervalles convergeant vers le zéro de φ s'il existe et d'intervalles vides à partir d'un indice k_0 si φ n'a pas de zéro sur $[x_0]$ ([Hansen92], [Neumaier90]). ♢

L'usage des intervalles étendus permet de recherche les zéros dans un intervalle où la dérivée s'annule. En utilisant (3.17) à l'itération k, on a

$$N\left([x]^k\right) = m\left([x]^k\right) - \frac{\varphi\left(m\left([x]^k\right)\right)}{\varphi'_{[]}\left([x]^k\right)} \tag{3.19}$$

or si $\varphi'_{[]}\left([x]^k\right)$ contient 0, dans le cas général et d'après les règles de calcul présentées au paragraphe 2.3, on a

$$N\left([x]^k\right) =]-\infty, \rho] \cup [\lambda, +\infty[= N_1\left([x]^k\right) \cup N_2\left([x]^k\right). \tag{3.20}$$

L'intersection de l'intervalle étendu $N\left([x]^k\right)$ avec $[x]^k$ peut être vide, dans ce cas, φ n'a pas de zéro sur $[x]^k$; elle pourra être un intervalle de plus petite longueur que $[x]^k$ ou encore générer deux intervalles distincts qu'il faudra examiner séparément.

La figure 3-6 donne une interprétation géométrique de la méthode de Newton par intervalles lorsque $0 \notin \varphi'([x])$. En effet, la formule (3.17) correspond à la recherche du point d'intersection x^* entre la droite $y = 0$ et les droites du faisceau de droites $y = \varphi(m([x])) + (x - m([x]))\,\varphi'([x])$, lorsque $[x]$ est fixé (en grisé sur la figure 3-6). Deux droites correspondant aux pentes minimales et maximales de φ sur $[x]$ sont tracées à partir du point $(m([x]), \varphi(m([x])))$. Ces deux droites coupent l'axe des abscisses en deux points définissant l'intervalle $N([x])$.

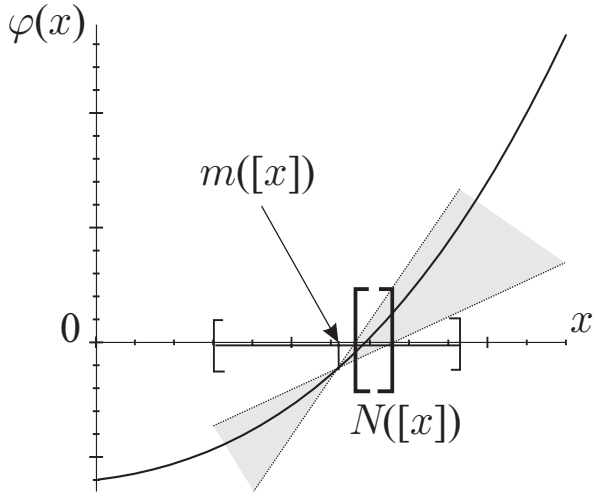


Figure 3-6: Algorithme de Newton par intervalles standard quand $0 \notin \varphi'([x])$.

Le cas où deux intervalles sont générés par itération de la méthode de Newton est illustré sur la figure 3-7. A nouveau, les droites de pente minimale et maximale sont tracées à partir du point $(m([x]), \varphi(m([x])))$ et coupent dans ce cas l'axe des abscisses en deux points définissant deux intervalles étendus $N_1([x])$ et $N_2([x])$, chacun susceptible de contenir un zéro.

Les propriétés de convergence de l'algorithme de Newton par intervalles peuvent être trouvées dans [Hansen92] ou [Neumaier90].

Nous nous intéressons à la recherche du minimum d'une fonction f ; rechercher les zéros de la dérivée de cette fonction permet d'obtenir à la fois les minima et les maxima de f. Une méthode de Newton par intervalles peut parfaitement être appliquée pour obtenir un encadrement affiné de ces zéros afin

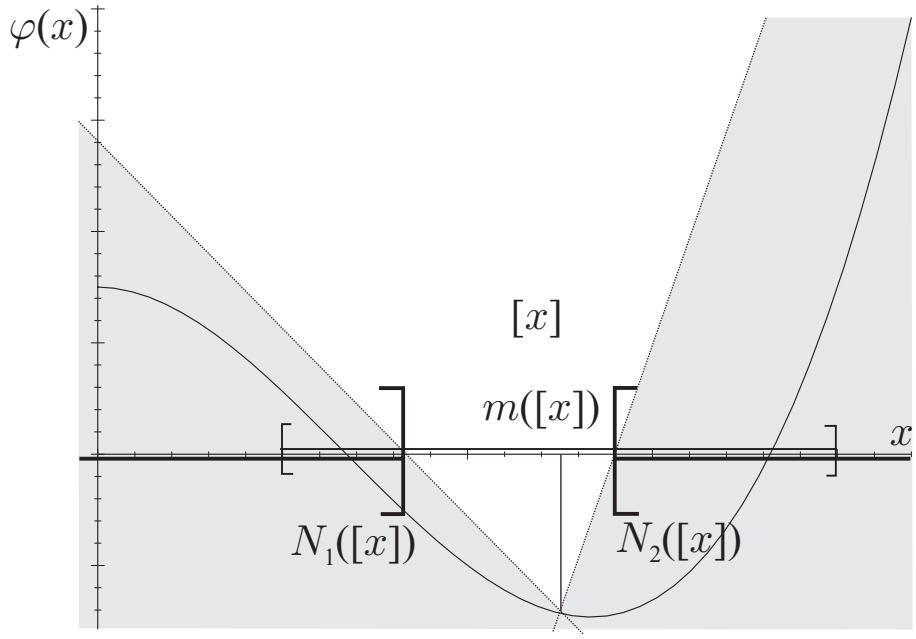


Figure 3-7: Algorithme de Newton par intervalles lorsque $0 \in \varphi'([x])$.

d'améliorer les encadrements des optimiseurs potentiels. Il faut cependant que la dérivée de la fonction à optimiser soit elle-même continûment dérivable pour appliquer cette méthode.

Extension : algorithme de Newton-Gauss-Seidel Dans le cas multidimentionnel, l'argument de l'optimum du critère à minimiser ne peut se trouver que dans un pavé où le gradient s'annule. La méthode de Newton-Gauss-Seidel, généralisation au cas multidimentionnel de la méthode de Newton par intervalles permet d'encadrer les zéros d'une fonction vectorielle, à condition que celle-ci soit différentiable. Comme la méthode de Newton, elle permet de réduire la taille des pavés de recherche des arguments de l'optimum global, elle peut éliminer certains pavés et isoler deux minimiseurs dans deux pavés distincts. En contrepartie, cette technique est beaucoup plus coûteuse en temps de calcul que les tests d'élimination. Pour plus de détails, voir [Alefeld et al.83] ou [Hansen et al.81].

Remarques complémentaires

Les pavés à traiter sont stockés dans une liste $\mathcal{L}$. A chaque pavé est associé son intervalle image par la fonction d'inclusion du critère. En fait, les pavés sont classés dans $\mathcal{L}$ par ordre croissant de la borne inférieure de cette évaluation. Cet agencement présente l'avantage de placer en tête de liste le pavé dont la borne inférieure est la plus faible, ce pavé étant jugé *a priori* le plus susceptible de contenir un minimiseur global. D'autre part, lorsque le majorant de l'optimum global est amélioré, il est possible d'effectuer un test du point milieu sur l'ensemble de la liste et d'éliminer tous les pavés dont la borne inférieure est supérieure au nouveau majorant. En commençant le test par la tête de liste, on peut, dès qu'un pavé ne

satisfait plus le test du point milieu, éliminer tous les pavés stockés après lui dans la liste sans faire de nouveaux tests.

Lorsqu'un pavé n'a pas pu être éliminé par l'ensemble des tests, il est coupé en deux afin d'obtenir une évaluation plus fine par la fonction d'inclusion du critère sur chacun des deux pavés ainsi obtenus. On peut choisir de bissecter le pavé dans son côté de plus grande longueur, ou de plus grande longueur relative. Ces deux stratégies sont simples, mais ne tiennent pas compte de la forme du critère à minimiser. Or, l'objectif étant d'isoler le plus rapidement possible des pavés où le critère est monotone, lorsque le gradient du critère est disponible, il peut être opportun de bissecter le pavé courant $[\mathsf{x}]$ suivant son coté d de plus grande longueur pondérée (voir p. 27)

$$d = \min \{ i = 1, \ldots, n \,|\, w\left([x]_i, p_i\right) = w\left([\mathsf{x}], \mathsf{p}\right) \} \tag{3.21}$$

avec p correspondant aux longueurs de chacune des composantes de l'évaluation intervalle du gradient de la fonction $f(x_1, \ldots, x_n)$ à optimiser

$$\mathsf{p} = \left(w\left(\left(\frac{\partial f}{\partial x_1} \right)_{[]} ([\mathsf{x}]) \right), \ldots, w\left(\left(\frac{\partial f}{\partial x_n} \right)_{[]} ([\mathsf{x}]) \right) \right)^{\mathrm{T}}. \tag{3.22}$$

Ceci permet de couper dans la direction où la variation du critère est la plus importante, et donc éventuellement de pouvoir isoler plus facilement des sous-pavés où le critère est monotone. L'exemple suivant illustre cette propriété.

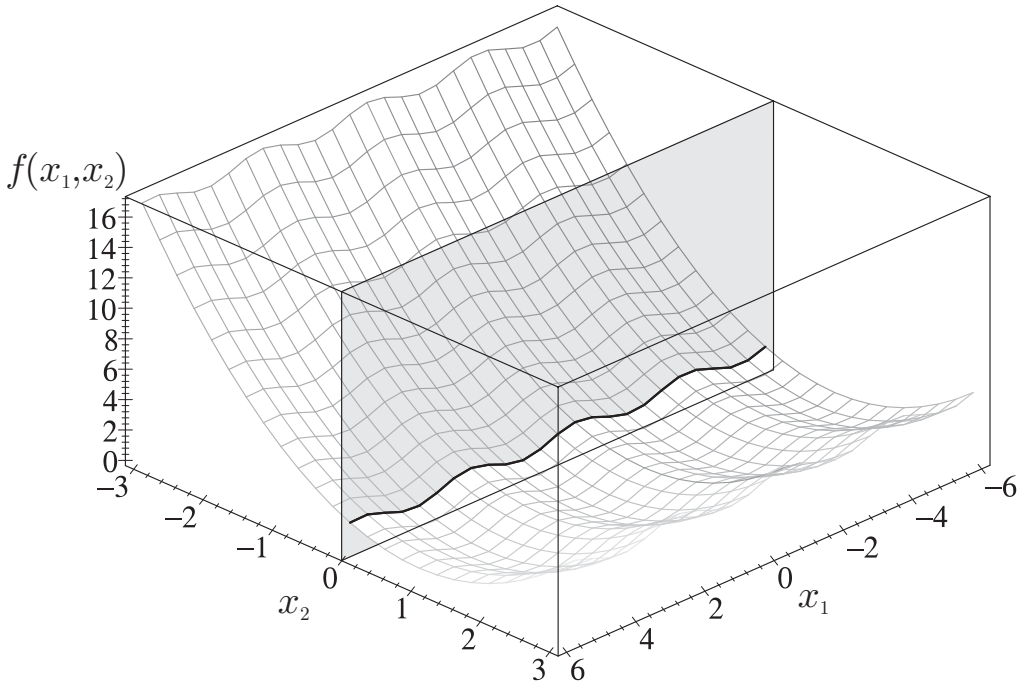


Figure 3-8: Représentation de $f(\mathsf{x}) = \sin^2(x_1 - 1) + (x_2 - 1)^2$ sur $[-6, 6] \times [-3, 3]$.

Exemple 3.2.2 *Considérons la fonction*

$$f(\mathsf{x}) = \sin^2(x_1 - 1) + (x_2 - 1)^2 \tag{3.23}$$

dont nous cherchons tous les arguments du minimum global sur le pavé $[\mathsf{x}] = [-6, 6] \times [-3, 3]$. *La représentation graphique de cette fonction est donnée sur la figure 3-8. Le gradient de* f *est*

$$\frac{\mathrm{d}f}{\mathrm{d}\mathsf{x}} = \begin{pmatrix} \sin(2x_1 - 2) \\ 2(x_2 - 1) \end{pmatrix} \tag{3.24}$$

et son Hessien est

$$\frac{\mathrm{d}^2 f}{\mathrm{d}\mathsf{x}^2} = \begin{pmatrix} 2\cos(2x_1 - 2) & 0 \\ 0 & 2 \end{pmatrix}. \tag{3.25}$$

On a $\left(\frac{\mathrm{d}f}{\mathrm{d}\mathsf{x}}\right)_{[]}([\mathsf{x}]) = \begin{pmatrix} [-1, 1] \\ [-8, 4] \end{pmatrix}$ *par conséquent, on ne peut pas éliminer ce pavé en utilisant le test de monotonicité. Comme* $\left(\frac{\mathrm{d}^2 f}{\mathrm{d}\mathsf{x}^2}\right)_{[]}([\mathsf{x}]) = \begin{pmatrix} [-1, 1] & 0 \\ 0 & 2 \end{pmatrix}$ *on ne peut non plus utiliser le test de convexité. La méthode de Newton-Gauss-Seidel permettrait sans doute d'obtenir des pavés coupés correctement, mais nous allons considérer pour cet exemple qu'elle n'est pas appliquée. Si une bissection est réalisée suivant la direction de plus grande longueur, pour les pavés obtenus* $[\mathsf{x}_1] = [-6, 0] \times [-3, 3]$ *et* $[\mathsf{x}_2] = [0, 6] \times [-3, 3]$, *il est aisé de vérifier que rien ne pourrait être conclu en utilisant les tests de monotonicité et de convexité. Par contre, en utilisant une bissection de la dimension de plus grande longueur pondérée par les longueurs des composantes du gradient, on obtient* $[\mathsf{x}'_1] = [-6, 6] \times [-3, 0]$ *et* $[\mathsf{x}'_2] = [-6, 6] \times [0, 3]$ *(voir le plan de coupe sur la figure 3-8). On a* $\left(\frac{\mathrm{d}f}{\mathrm{d}\mathsf{x}}\right)_{[]}([\mathsf{x}'_1]) = \begin{pmatrix} [-1, 1] \\ [-8, -2] \end{pmatrix}$, *la fonction est donc strictement monotone sur* $[\mathsf{x}'_1]$ *et ce pavé peut être éliminé de la liste des pavés contenant les arguments potentiels de l'optimum.* ♢

3.3 Application à l'identification de modèles compartimentaux

Nous abordons dans cette partie le problème de l'identification des paramètres de modèles dont la sortie est constituée d'une somme d'exponentielles

$$y_{\mathrm{m}}(t_k, p) = \sum_{i=1}^{n} a_i . \exp(-b_i . t_k). \tag{3.26}$$

Les paramètres à identifier sont les a_i et b_i ; les t_k représentant les instants de mesure.

De telles sorties se rencontrent par exemple lors de l'étude de modèles compartimentaux. Ces modèles

ont la propriété de présenter des sorties dont l'évolution est dans la plupart des cas non oscillante, lorsqu'on excite les entrées par des impulsions ou des échelons, contrairement à ce que l'on peut observer lorsqu'on étudie la réponse de certains circuits électriques linéaires.

Dans un premier temps, nous présenterons brièvement le concept de modèle compartimental et ferons le lien entre ce type de représentation et la représentation d'état, plus familière aux automaticiens. Ensuite, nous étudierons un exemple d'identification d'un modèle à deux compartiments.

3.3.1 Modèles compartimentaux

Ce paragraphe reprend une partie de la présentation faite dans [Godfrey83] ; pour de plus amples informations concernant les modèles compartimentaux, on pourra consulter cet ouvrage ou [Jacquez72].

Un système compartimental est constitué d'un ensemble fini de sous-systèmes, homogènes et disjoints, appelés compartiments. Ces derniers échangent de la matière entre eux et avec l'extérieur. Au sein d'un compartiment, l'évolution de la quantité de matière est supposée décrite par une équation différentielle du premier ordre.

Les modèles compartimentaux sont souvent utilisés pour décrire l'évolution de la localisation d'une ou plusieurs substances dans un organisme. Dans ce cas, chaque compartiment correspond à un site différent. On pourra également utiliser les compartiments pour représenter différents composants (médicaments ou métabolites par exemple), les échanges entre compartiments pourront alors correspondre à des transformations chimiques et plusieurs compartiments pourront partager un même site.

La figure 3-9 illustre les échanges entre deux compartiments d'un modèle compartimental et le milieu extérieur.

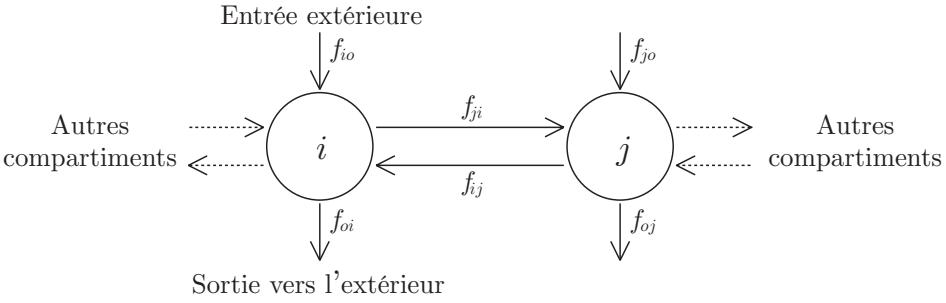


Figure 3-9: Modèles compartimentaux.

Les différents débits de matière f peuvent être représentés par une fonction quelconque positive. f_{ji} représente le débit de matière vers le compartiment j à partir du compartiment i ; f_{oi} et f_{io} représentent les échanges avec le milieu extérieur. Lorsque les f_{oi} sont tous nuls, le système est dit *fermé* : il n'expulse pas de matière vers l'extérieur.

L'évolution de la quantité de matière x_i au sein du compartiment i, élément d'un modèle à n com-

partiments, suit donc la loi de conservation

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = f_{io} + \sum_{j=1, j\neq i}^{n} f_{ij} - \sum_{j=1, j\neq i}^{n} f_{ji} - f_{oi}. \tag{3.27}$$

Cette structure est d'un intérêt limité, car elle n'est guère exploitable sans spécifier les lois correspondant aux f_{ij}. Nous considérerons des systèmes faisant intervenir des débits de matière linéaires et invariants, ce qui correspond à des équations différentielles linéaires stationnaires. Ce cas de figure est relativement fréquent dans les études pratiques. Il correspond d'une part à la linéarisation de systèmes non linéaires, mais également permet de traduire l'évolution de la proportion d'un marqueur (par exemple une substance radioactive) au sein d'un système quelconque ayant atteint un régime permanent. On se réfèrera à [Walter82], pp. 8 à 18, pour une présentation détaillée de la linéarisation par inclusion d'un marqueur.

Dans le cas linéaire et invariant, l'évolution de la quantité de matière (3.27) prend la forme simplifiée

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = \sum_{j=1, j\neq i}^{n} k_{ij}.x_j - \sum_{j=1, j\neq i}^{n} k_{ji}.x_i - k_{oi}.x_i + f_{io}. \tag{3.28}$$

Par convention, les flux de matière sont toujours positifs, les k_{ij} sont donc tous des réels positifs ou nuls. De plus, si on pose $f_{io} = u_i(t)$, où $u_i(t)$ représente l'entrée de matière dans le compartiment i depuis le milieu extérieur, on peut transformer la modélisation compartimentale en représentation d'état

$$\dot{\mathsf{x}} = \mathsf{A}.\mathsf{x} + \mathsf{B}.\mathsf{u}, \tag{3.29}$$

avec

$$\begin{aligned} a_{ij} &= k_{ij}, \text{ si } i \neq j, \\ a_{ii} &= -\sum_{j=1, j\neq i}^{n} k_{ji} - k_{oi}. \end{aligned} \tag{3.30}$$

Pour une entrée $u_i(t)$ impulsionnelle, la quantité de matière x_i dans les différents compartiments suit une évolution caractérisée par une somme d'exponentielles le plus souvent réelles. Godfrey [Godfrey83] montre que dans le cas général, la sortie comporte n exponentielles lorsque le modèle est à n compartiments, et $n-1$ exponentielles lorsque ce système est fermé (présence d'un intégrateur pur).

D'autre part, Goldberg [Goldberg56] montre que les arguments des exponentielles de la somme sont réels, si pour tous les cycles comportant r compartiments du modèle, un cycle étant un ensemble de compartiments interconnectés $\alpha \to \beta \to ... \to \rho \to \alpha$ (voir la figure 3-10), on a

$$k_{\alpha\beta}k_{\beta\gamma}...k_{\rho\alpha} = k_{\beta\alpha}k_{\gamma\beta}...k_{\alpha\rho}. \tag{3.31}$$

Cette propriété est particulièrement intéressante dans le cas d'un modèle à deux compartiments : si un

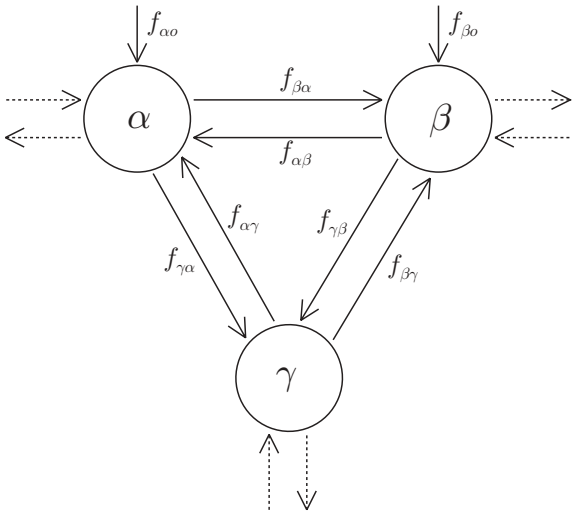


Figure 3-10: Modèle compartimental présentant un cycle $\alpha \rightarrow \beta \rightarrow \gamma$.

cycle est présent, nécessairement, on a

$$k_{12}.k_{21} = k_{21}.k_{12}, \tag{3.32}$$

et la sortie est constituée d'une somme d'exponentielles réelles, quelle que soit la valeur des micropa-ramètres k_{ij}.

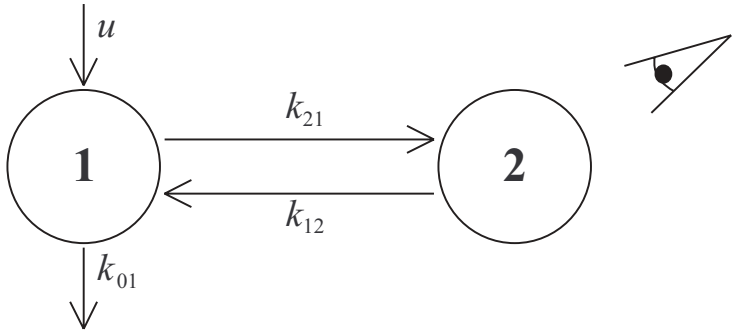


Figure 3-11: Modèle à deux compartiments.

La figure 3-11 présente un modèle à deux compartiments qui servira de support pour la suite de cette application. Considérons $\mathsf{x} = (x_1, x_2)^{\mathrm{T}}$, le vecteur des quantités de matière dans chacun des deux compartiments. Son évolution est décrite par le système suivant

$$\begin{cases} \dot{x}_1 = -\left(k_{21} + k_{01}\right) x_1 + k_{12} x_2 + \ u, \\ \dot{x}_2 = k_{21} x_1 - k_{12} x_2. \end{cases} \tag{3.33}$$

Supposons que seul le compartiment 2 est observé, qu'une impulsion de Dirac unité est appliquée en entrée ($u(t) = \delta(t)$), que l'état initial du système est $\mathsf{x_0} = (0,0)^{\mathrm{T}}$ et que l'équation d'observation est la suivante

$$y_{\mathrm{s}}(t_i) = 10x_2\,(t_i) + \; b(t_i), \quad i = 1, ..., 16 \tag{3.34}$$

où les $b(t_i)$ sont les réalisations de variables gaussiennes indépendantes, de moyenne nulle et identiquement distribuées. Les données ont été obtenues en ajoutant un bruit pseudo-aléatoire à des données simulées et non bruitées ; en outre, une troncature à la troisième décimale a été effectuée (voir le tableau 3.1).

t_i	1	2	3	4	5	6	7	8
$y\,(t_i)$	0.532	0.478	0.410	0.328	0.323	0.148	0.216	0.127
t_i	9	10	11	12	13	14	15	16
$y\,(t_i)$	0.099	0.081	0.065	0.043	0.013	0.015	0.060	0.126

Tableau 3.1: Données simulées.

On peut aisément montrer (voir [Godfrey83]) que les observations satisfont

$$y_{\mathrm{s}}\,(t_i) = \alpha^* \left(\mathrm{e}^{\lambda_1^* t_{\mathrm{i}}} - \mathrm{e}^{\lambda_2^* t_{\mathrm{i}}}\right) + \; b\,(t_i)\,, \quad i = 1, ..., 16, \tag{3.35}$$

avec

$$\begin{aligned} \alpha^* &= \frac{10k_{21}^*}{\sqrt{\left(k_{01}^* - k_{12}^* + k_{21}^*\right)^2 + 4k_{12}^* k_{21}^*}}, \\ \lambda_{1,2} &= -\tfrac{1}{2}\left[(k_{01}^* + k_{12}^* + k_{21}^*) \pm \sqrt{(k_{01}^* - k_{12}^* + k_{21}^*)^2 + 4k_{12}^* k_{21}^*}\right], \end{aligned} \tag{3.36}$$

et où l'étoile (*) en exposant signifie qu'il s'agit de la vraie valeur des paramètres. Il est ensuite possible d'estimer les *microparamètres* $\mathsf{p}_\mu = (k_{12}, k_{21}, k_{01})$ ou les *macroparamètres* $\mathsf{p}_{\mathrm{m}} = (\alpha, \lambda_1, \lambda_2)$ du modèle compartimental. Dans les deux cas, l'estimée au sens du maximum de vraisemblance est obtenue en minimisant la fonction de coût

$$j(\mathsf{p}) = \sum_{i=1}^{16} \left[y_{\mathrm{s}}\,(t_i) - y_{\mathrm{m}}\,(t_i, \mathsf{p})\right]^2, \tag{3.37}$$

où $y_{\mathrm{m}}(t_i, \mathsf{p})$ est la partie déterministe du modèle et où p est le vecteur des macro ou des microparamètres.

3.3.2 Estimation des macroparamètres

La partie déterministe de la sortie du modèle peut être exprimée en fonction des macroparamètres $\mathsf{p}_\mathrm{m} = (\alpha, \lambda_1, \lambda_2)$ de la manière suivante

$$y_\mathrm{m}(t_i, \mathsf{p}_\mathrm{m}) = \alpha \left(\mathrm{e}^{\lambda_1 t_\mathrm{i}} - \mathrm{e}^{\lambda_2 t_\mathrm{i}} \right). \tag{3.38}$$

Un pavé $[\hat{\mathsf{p}}_\mathrm{m}]$ encadrant l'estimée au sens du maximum de vraisemblance de ces macroparamètres est obtenue en minimisant la fonction de coût correspondante $j(\mathsf{p}_\mathrm{m})$. Ceci est réalisé par l'algorithme d'optimisation globale garantie décrit dans la section 3.2.2. L'intervalle de recherche choisi est $[0,\ 2.5] \times [-5,\ 0] \times [-5,\ 0]$ le paramètre de précision ϵ a été fixé à 10^{-9}. Deux méthodes de bissection des pavés sont utilisées (voir le paragraphe 2.6). Quelle que soit la méthode employée, l'algorithme fournit un pavé solution unique

$$\begin{aligned} [\hat{\mathsf{p}}_\mathrm{m}] \quad &= \quad [\alpha] \times [\lambda_1] \times [\lambda_2] = [0.7736908, 0.7736912] \\ &\quad \times [-0.2146864, -0.2146863] \times [-2.093262, -2.093260]. \end{aligned}$$

Ce pavé a été stocké dans la liste solution car le diamètre de l'évaluation intervalle du critère sur $[\hat{\mathsf{p}}_m]$ est plus petit que ϵ. Pendant la recherche, deux minima locaux ont été trouvés, et l'algorithme a prouvé qu'ils n'étaient pas globaux. Les calculs effectués sur un Pentium 233MMX sont résumés dans le tableau 3.2.

Méthode de bissection	Nb d'itérations	Temps de calcul
Diamètre maximal	7260	433 s
Diam. max. pondéré	4964	282 s

Tableau 3.2: Algorithme d'optimisation, comparaison des méthodes de bisection.

Le poids utilisé pour la pondération pour la seconde méthode est celui décrit par la formule (3.21), et le nombre d'itérations correspond au nombre de fois que l'étape "extraire $[\mathsf{x}]$ de $\mathcal{L}$" est effectué dans l'algorithme de la figure 3-2. Nous constatons que la seconde méthode est environ 50% plus rapide. L'utilisation du poids (3.21) permet bien d'isoler plus rapidement les pavés où le critère est monotone et donc de les éliminer plus rapidement.

La figure 3-12 illustre les données mesurées $y_\mathrm{s}(t_i)$ et les courbes correspondant à la sortie du modèle $y_\mathrm{m}(t, \mathsf{p}_\mathrm{m})$ pour ces deux optima locaux et pour l'optimum global.

Pourvu qu'il contienne le minimiseur recherché, la taille de l'espace de recherche initial ne joue pas un rôle essentiel : lorsque la taille de l'espace de recherche augmente, la solution reste la même, mais le temps de calcul augmente. Par exemple, lorsque le pavé de recherche initial est $[0,\ 5] \times [-5,\ 0] \times [-5,\ 0]$, une solution identique est trouvée en 800 s (en utilisant la méthode de bissection suivant le diamètre maximal pondéré). Les bornes du pavé de recherche sont le plus souvent déterminées par des considérations physiques sur les grandeurs à estimer.

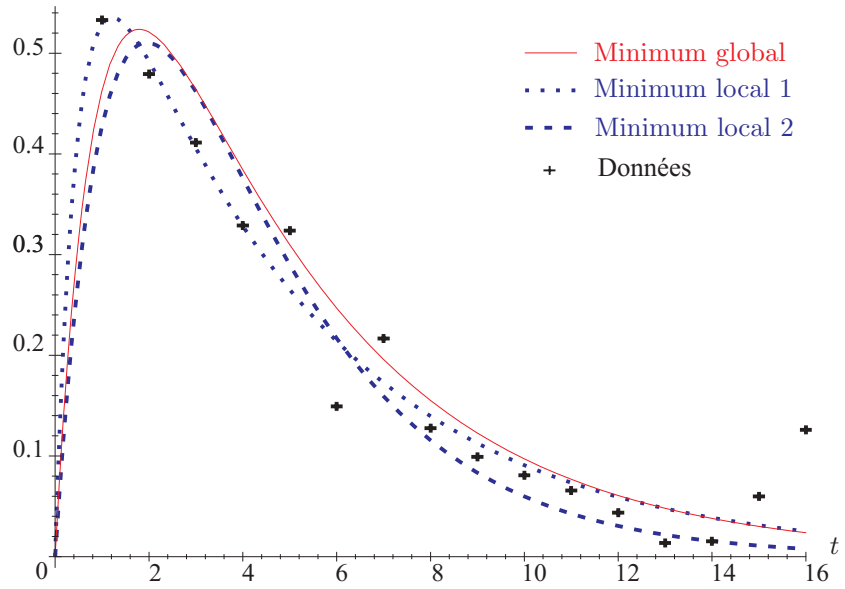


Figure 3-12: Données et sorties des modèles.

3.3.3 Estimation des microparamètres à partir des macroparamètres

Le vecteur de macroparamètres p_{m} peut être exprimé en fonction des microparamètres $\mathsf{p}_\mu = (k_{21}, k_{12}, k_{01})$ à l'aide de la formule (3.36). Bien qu'il soit aussi possible d'expliciter tous les vecteurs de microparamètres compatibles avec p_{m}, nous avons choisi de résoudre (3.36) de manière garantie à l'aide d'une technique d'analyse par intervalles. On se reportera par exemple dans [Neumaier90] ou [Hammer et al.95] pour plus de détails. La procédure utilisée, met en œuvre la méthode de Newton-Gauss-Seidel et fournit un encadrement de toutes les solutions du système incluse dans un pavé de recherche de départ.

Nous avons choisi le pavé de recherche $[0.01, 2.0] \times [0.05, 3.0] \times [0.05, 3.0]$ pour p_μ. Seules les solutions fournissant des microparamètres positifs sont retenues, car ce sont les seules qui correspondent à des modèles compartimentaux. Les deux pavés contenant l'estimée des microparamètres associés à l'estimée des macroparamètres au sens du maximum de vraisemblance ont été obtenus en moins d'une seconde et sont

$$\begin{aligned}
[\hat{\mathsf{p}}_\mu]_1 &= [k_{21}]_1 \times [k_{12}]_1 \times [k_{01}]_1 = [0.1450762, 0.1450766] \\
&\quad \times [1.925402, 1.925404] \times [0.2327170, 0.2327181]\,, \\
[\hat{\mathsf{p}}_\mu]_2 &= [k_{21}]_2 \times [k_{12}]_2 \times [k_{01}]_2 = [0.1450762, 0.1450766] \\
&\quad \times [0.2327170, 0.2327181] \times [1.925402, 1.925404]\,.
\end{aligned}$$

Une étude d'identifiabilité [Walter82] confirmerait que les paramètres k_{01} et k_{12} sont seulement localement identifiables et peuvent être interchangés, le système conservant le même comportement entrée-sortie.

Un calcul de la valeur des microparamètres que l'on peut obtenir à partir de la valeur des macropa-

ramètres correspondant aux deux minima locaux est également réalisé. Seul le premier correspond à un modèle comportemental. La figure 3-13 résume les résultats obtenus (tronqués à la troisième décimale).

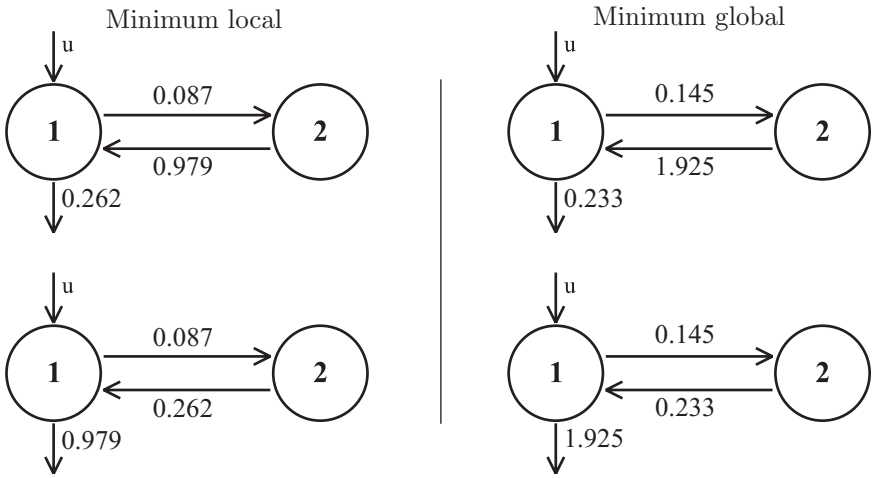


Figure 3-13: Résultats de l'estimation des microparamètres.

Si une recherche locale avait été entreprise, l'algorithme classique employé aurait pu fournir une estimée correspondant à un minimum local du critère, la valeur des microparamètres étant alors relativement éloignée de celle obtenue pour l'argument de l'optimum global.

3.3.4 Estimation directe des microparamètres

Les microparamètres peuvent également être estimés par minimisation directe de la fonction de coût $j(\mathsf{p}_\mu)$. $j(\mathsf{p}_\mathrm{m})$ est obtenu en remplaçant p_m par son expression en fonction de $\mathsf{p}_\mu = (k_{12}, k_{21}, k_{01})$.

L'intervalle de recherche est $[0.01, 2.0] \times [0.05, 3.0] \times [0.05, 3.0]$; le paramètre de précision ϵ a été pris à 10^{-9}. La recherche prend alors près d'un jour sur le même ordinateur. Deux pavés sont obtenus :

$$\begin{aligned}
[\hat{\mathsf{p}}_\mu]_1 &= [0.145075, 0.145077] \\
&\quad \times [1.925402, 1.925404] \times [0.232717, 0.232719], \\
[\hat{\mathsf{p}}_\mu]_2 &= [0.145075, 0.145077] \\
&\quad \times [0.232717, 0.232719] \times [1.925402, 1.925404].
\end{aligned}$$

Ces pavés contiennent les solutions obtenues à la section 3.3.3.

Notons qu'il n'a pas été nécessaire de fournir à l'algorithme d'expression explicite du gradient et du Hezssien du critère, ceux-ci étant obtenus par différentiation automatique (voir [Rall81] ou [Hammer et al.95]) pendant l'exécution du programme de recherche.

Le temps de calcul beaucoup plus important est du à la plus grande complexité de la fonction d'inclusion. Les variables intervenant avec plus d'une occurrence dans la fonction de coût, la fonction d'inclusion

obtenue n'est pas minimale (cf. paragraphe 2.5), et un découpage plus important doit être effectué pour éliminer des pavés de l'espace de recherche.

3.4 Contexte erreurs bornées

Si, dans certains cas, l'hypothèse d'un bruit gaussien sur les erreurs est raisonnable, dans de nombreuses autres circonstances, elle ne trouve pas de réelle justification. En effet, lorsqu'on considère par exemple une erreur de modèle, celle-ci sera reproductible et ne dépendra pas des circonstances de l'expérimentation. D'autre part, lorsqu'un appareil de mesure de calibre donné est utilisé, la mesure fournie se trouve à l'intérieur d'un intervalle déterminé par le calibre. Aussi, allons-nous étudier dans ce paragraphe l'estimation paramétrique dans le contexte d'erreurs bornées.

Nous supposerons que pour tout t_i, et connaissant la mesure $y_{\mathrm{s}}(t_i)$, il est possible de définir un intervalle $[y_{\mathrm{s}}(t_i)]$ satisfaisant

$$y_{\mathrm{s}}(t_i) \in [y_{\mathrm{s}}(t_i)] \tag{3.39}$$

et tel que $[y^{\mathrm{s}}(t_i)]$ contienne de manière *garantie* la valeur de la sortie à l'instant t_i si l'expérience est répétée. Identifier les paramètres d'un modèle $y_{\mathrm{m}}(t_i, \mathsf{p})$ d'un système ayant pour sorties mesurées $y_{\mathrm{s}}(t_i)$ revient à trouver l'ensemble des valeurs de p satisfaisant

$$y_{\mathrm{m}}(t_i, \mathsf{p}) \in [y_{\mathrm{s}}(t_i)] \text{ pour } i = 1, \ldots, n. \tag{3.40}$$

L'erreur entre la valeur mesurée de la sortie et la valeur estimée par le modèle est alors bornée :

$$\begin{array}{rcl} e(t_i, \mathsf{p}) & = & y_{\mathrm{s}}(t_i) - y_{\mathrm{m}}(t_i, \mathsf{p}) \in y_{\mathrm{s}}(t_i) - [y_{\mathrm{s}}(t_i)]\,. \\ e(t_i, \mathsf{p}) & \in & \left[y_{\mathrm{s}}(t_i) - \overline{y_{\mathrm{s}}(t_i)}, y^{\mathrm{s}}(t_i) - \underline{y_{\mathrm{s}}(t_i)}\right] = [e(t_i)]\,. \end{array} \tag{3.41}$$

Ainsi, pour $i = 1, \ldots, n$, lorsque $y_{\mathrm{s}}(t_i)$ et $[y_{\mathrm{s}}(t_i)]$ sont connus, on a $[e(t_i)] = \left[y_{\mathrm{s}}(t_i) - \overline{y_{\mathrm{s}}(t_i)}, y^{\mathrm{s}}(t_i) - \underline{y_{\mathrm{s}}(t_i)}\right]$. Rechercher l'ensemble des valeurs des paramètres *compatibles* avec les mesures et les bornes des erreurs de mesures revient à caractériser

$$\mathcal{S} = \{\mathsf{p} \in \mathcal{P} \,|\, y_{\mathrm{m}}(t_i, \mathsf{p}) \in [y_{\mathrm{s}}(t_i)]\,, i = 1, \ldots, n\} = \{\mathsf{p} \in \mathcal{P} \,|\, \mathsf{y}^{\mathrm{m}}(\mathsf{p}) \in [\mathsf{y}^{\mathrm{s}}]\}\,. \tag{3.42}$$

On peut également définir $\mathcal{S}$ de manière équivalente à partir de l'erreur $e(t_i, \mathsf{p})$:

$$\mathcal{S} = \{\mathsf{p} \in \mathcal{P} \,|\, e(t_i, \mathsf{p}) \in [e(t_i)]\,, i = 1, \ldots, n\} = \{\mathsf{p} \in \mathcal{P} \,|\, \mathsf{e}(\mathsf{p}) \in [\mathsf{e}]\}\,. \tag{3.43}$$

$\mathcal{S}$ n'est en général pas un singleton. Il peut être vide, ce qui signifie qu'il n'est pas possible de trouver de valeurs de p satisfaisant (3.40) pour tout i, ou encore être un sous-ensemble de $\mathcal{P}$, éventuellement non

connexe. Dans ce dernier cas, plusieurs valeurs de paramètres, différentes les unes des autres, conduisent à un modèle fournissant des résultats proches des données mesurées.

Diverses techniques ont été développées pour caractériser l'ensemble $\mathcal{S}$, et [Wal90], [Nor94], [Nor95] ou encore [Ple96] permettent d'avoir un aperçu des méthodes de résolution disponibles, qui diffèrent suivant que la sortie du modèle est linéaire en les paramètres ou pas.

La technique présentée ici a été introduite parallèlement dans [Moore92] et [Jaulin et al.93]. Elle permet de caractériser de manière approchée, mais avec une précision arbitraire, l'ensemble $\mathcal{S}$ par un *sous-pavage*, c'est-à-dire par un ensemble constitué de l'union de pavés disjoints. Ici, une version récursive de l'algorithme proposé dans [Jaulin et al.93], plus efficace que la version originale, sera présentée. En outre, les notions de tests d'inclusion et de masques permettront également d'améliorer ses performances.

3.4.1 Inversion ensembliste - Sivia

La caractérisation de l'ensemble $\mathcal{S}$ peut être interprétée comme un problème d'*inversion ensembliste*. On peut considérer $\mathcal{S}$ comme l'image d'un ensemble par une fonction réciproque (au sens ensembliste), en effet, (3.42) peut également s'écrire

$$\mathcal{S} = (\mathsf{y}^{\mathrm{m}})^{-1}([\mathsf{y}^{\mathrm{s}}]) \cap \mathcal{P}, \tag{3.44}$$

qui sera noté $(\mathsf{y}^{\mathrm{m}})^{-1}_{\mathcal{P}}([\mathsf{y}^{\mathrm{s}}])$, et d'après (3.43), on a

$$\mathcal{S} = \mathsf{e}^{-1}([\mathsf{e}]) \cap \mathcal{P}, \tag{3.45}$$

noté $\mathsf{e}^{-1}_{\mathcal{P}}([\mathsf{e}])$.

Le problème d'inversion ensembliste le plus général est formulé de la manière suivante : soit $\mathcal{P} \subset \mathbf{R}^n$ et $\mathcal{E} \subset \mathbf{R}^m$ (ou $\mathbf{B}^m$) et une fonction $\mathsf{f} : \mathbf{R}^n \to \mathbf{R}^m$ (ou $\mathbf{B}^m$). Décrire l'ensemble

$$\mathcal{S} = \{\mathsf{p} \in \mathcal{P} \,|\, \mathsf{f}(\mathsf{p}) \in \mathcal{E}\} \tag{3.46}$$

revient à caractériser

$$\mathcal{S} = \mathsf{f}^{-1}_{\mathcal{P}}(\mathcal{E}). \tag{3.47}$$

L'analyse par intervalles permet de fournir une réponse approchée mais garantie à ce problème grâce aux fonctions d'inclusion. Considérons le problème de la caractérisation de l'ensemble $\mathcal{S}$ décrit par les équations (3.46) ou (3.47). Supposons qu'une fonction d'inclusion convergente $\mathsf{f}_{[]}(.)$ définie sur l'ensemble des intervalles de $\mathcal{P}$ soit disponible pour $\mathsf{f}(.)$ et considérons un pavé $[\mathsf{p}] \subset \mathcal{P}$. Si $\mathsf{f}_{[]}([\mathsf{p}]) \subset \mathcal{E}$, alors pour tout $\mathsf{p} \in [\mathsf{p}], \mathsf{f}(\mathsf{p}) \in \mathcal{E}$ et $[\mathsf{p}] \subset \mathcal{S}$, le pavé $[\mathsf{p}]$ est dit *admissible*. Si $\mathsf{f}_{[]}([\mathsf{p}]) \cap \mathcal{E} = \emptyset$, alors pour tout $\mathsf{p} \in [\mathsf{p}], \mathsf{f}(\mathsf{p}) \notin \mathcal{E}$ et $[\mathsf{p}] \cap \mathcal{S} = \emptyset$, le pavé $[\mathsf{p}]$ est dit *inadmissible*. Enfin, lorsque les deux premières

conditions ne sont pas satisfaites, une partie de $[\mathbf{p}]$ peut éventuellement appartenir à $\mathcal{S}$, $[\mathbf{p}]$ est alors dit *incertain*.

Exemple 3.4.1 *Considérons une fonction* $\mathbf{f} : \mathbf{R}^2 \to \mathbf{R}^2$, *un pavé* $\mathcal{P}$ *de* $\mathbf{R}^2$ *et* $\mathcal{E} = (\mathbf{R}^+)^2$. *On suppose qu'une fonction d'inclusion* $\mathbf{f}_{[]}$ *est disponible pour* $\mathbf{f}$. *Il s'agit de caractériser* $\mathcal{S} = \mathbf{f}_{\mathcal{P}}^{-1}(\mathcal{E})$. *La figure 3-14 illustre les différents type de pavés. Le pavé* 1 *est admissible car son image par la fonction d'inclusion est incluse dans* $\mathcal{E}$. *Le pavé* 2 *est inadmissible car son image par la fonction d'inclusion est à l'extérieur de* $\mathcal{E}$. *Le grand pavé* 3 *contenant à la fois les pavés* 1 *et* 2 *est incertain car son image par la fonction d'inclusion a une intersection non vide avec* $\mathcal{E}$; *ce résultat n'est pas surprenant car le pavé* 3 *contient des pavés admissibles et inadmissibles.* ♢

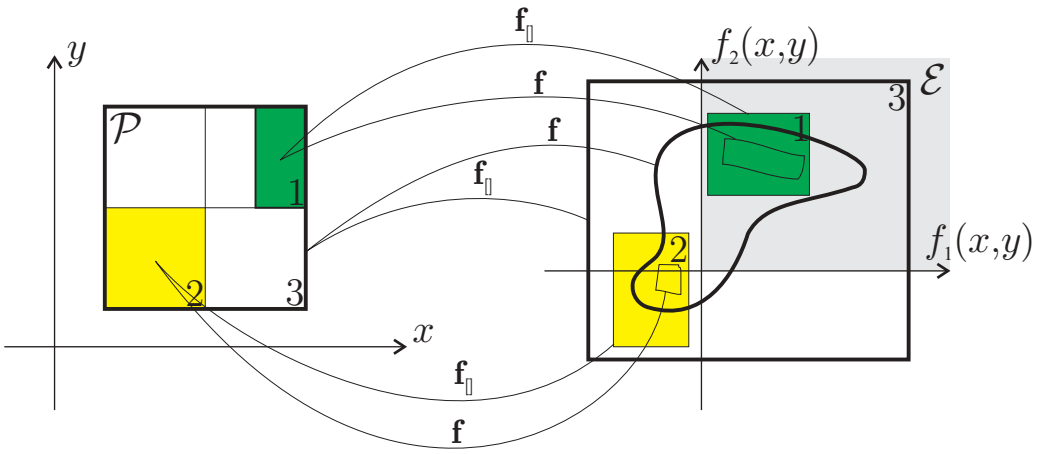


Figure 3-14: Différents types de pavés sur un problème d'inversion de $\mathcal{E} = [0, +\infty[^2$.

L'algorithme Sivia (*Set Inversion Via Interval Analysis*, inversion ensembliste utilisant l'analyse par intervalle) met en œuvre la propriété de convergence de la fonction d'inclusion utilisée afin de caractériser de manière récursive l'ensemble $\mathcal{S}$. Un pavé $[\mathbf{p}] \subset \mathcal{P}$ est considéré. S'il est admissible, il est stocké dans l'ensemble $\mathcal{S}^-$ des pavés admissibles. S'il est inadmissible, il est rejeté. Enfin, s'il est incertain, à condition que sa longueur maximale (éventuellement relative ou pondérée) soit supérieure à un paramètre de précision ϵ, il est bissecté en utilisant l'une des règles canoniques définie au chapitre 2. Le test est alors appliqué récursivement sur les deux sous-pavés obtenus. Par contre, si $[\mathbf{p}]$ est trop petit pour être encore bissecté (par exemple, si sa plus grande longueur est inférieure au paramètre de précision ϵ), il est stocké dans l'ensemble $\mathcal{S}^0$ des pavés incertains. Cet algorithme est fini, sa complexité (exponentielle) a été étudiée dans [Jaulin et al.93]. En fin d'exécution, on dispose de deux ensembles $\mathcal{S}^-$ et $\mathcal{S}^0$ et on garantit que

$$\mathcal{S}^- \subset \mathcal{S} \subset \left(\mathcal{S}^- \cup \mathcal{S}^0\right). \tag{3.48}$$

En utilisant les notations précédentes, la syntaxe de Sivia est

$$\left(\mathcal{S}^{-},\mathcal{S}^{0}\right)=\mathsf{Sivia}\left(\mathcal{P},\mathsf{f},\mathcal{E},\epsilon\right). \tag{3.49}$$

Remarque 3.4.1 *Dans ce chapitre, nous supposerons que l'ensemble $\mathcal{P}$ de recherche de $\mathcal{S}$ est un pavé noté $[\mathsf{p}_0]$ et que l'ensemble $\mathcal{E}$ est un pavé éventuellement infini. En effet, il est alors très facile de déterminer si deux pavés sont disjoints, inclus l'un dans l'autre ou s'ils ont une intersection non vide ; par contre, déterminer si un pavé est inclus dans un ensemble quelconque est beaucoup plus délicat et nécessite une représentation structurée de l'ensemble en question. Cette représentation structurée sera abordée au chapitre 5. Nous verrons alors qu'il est possible de rechercher $\mathcal{S}$ dans un ensemble bien plus complexe qu'un simple pavé.* ♢

3.4.2 Inversion d'ensembles définis par des tests booléens

Au chapitre 4 et dans de nombreuses autres situations, l'ensemble à inverser sera le singleton $\{1\}$ correspondant à la valeur booléenne *vrai*. En effet, souvent il faut trouver l'ensemble des paramètres d'un modèle satisfaisant une condition logique. Considérons donc le cas où la fonction intervenant dans la définition de $\mathcal{S}$ est un test booléen.

Remarquons que tous les problèmes d'inversion ensembliste peuvent se mettre sous cette forme ; on peut par exemple réécrire (3.46) sous la forme

$$\mathcal{S}=\{\mathsf{p}\in\mathcal{P}\,|\,t\left(\mathsf{p}\right)=1\}, \tag{3.50}$$

avec

$$t\left(\mathsf{p}\right)=1 \text{ équivaut à } \mathsf{f}\left(\mathsf{p}\right)\in\mathcal{E} \tag{3.51}$$

ou de manière équivalente

$$\mathcal{S}=t_{\mathcal{P}}^{-1}\left(1\right). \tag{3.52}$$

Dans ce cas, l'algorithme d'inversion ensembliste mettra en œuvre un test d'inclusion $t_{[]}\left(.\right)$ du test t, mais la structure de Sivia restera celle qui a été décrite par exemple dans [Jaulin et al.93]. Cependant, une formulation du problème sous forme de test permet d'accélérer fortement l'algorithme à l'aide d'un vecteur de *masques* [Kieffer et al.99a].

Supposons que $t\left(\mathsf{p}\right)$ soit la combinaison de n tests élémentaires t_i sur le pavé $[\mathsf{p}]$. Lorsque la valeur du test d'inclusion $t_{i[]}$ associé à t_i est soit vrai, soit faux sur $[\mathsf{p}]$, le résultat obtenu reste le même sur tout sous-pavé de $[\mathsf{p}]$. De ce fait, il n'est pas indispensable d'évaluer à nouveau $t_{i[]}$ sur les sous-pavés de $[\mathsf{p}]$, et seuls les tests élémentaires dont le résultat était indéterminé doivent être réévalués. L'utilisation d'un

vecteur de masques, stockant la valeur des tests élémentaires permet de réaliser cette opération (voir par exemple [Ratschek et al.88], pp. 165-168). Le masque associé au test d'inclusion $t_{[]}$ pour le pavé $[\mathsf{p}]$ est la fonction $\Pi_{[]}(.)$ à valeur dans $\mathbf{IB}^n$ définie par

$$\Pi_{[]}([\mathsf{p}]) = \left(t_{1[]}([\mathsf{p}]), \ldots, t_{n[]}([\mathsf{p}])\right)^{\mathrm{T}}. \tag{3.53}$$

Sauf lorsque $[\mathsf{p}] = [\mathsf{p}_0]$, les résultats des tests élémentaires $t_{i[]}$ sur $[\mathsf{p}]$ seront déjà disponibles pour le pavé père dont $[\mathsf{p}]$ résulte par bissection. Si ces résultats ont été stockés dans un vecteur de masque $[\Pi]$ attaché au pavé père, il suffit alors d'évaluer les tests élémentaires qui ont obtenu un résultat incertain. La version utilisant un masque $[\Pi]$ du test d'inclusion $t_{[]}$ sera notée $t_{[]}([\mathsf{p}], [\Pi])$. Le masque attaché à $[\mathsf{p}]$ sera déduit du masque $[\Pi]$; certaines composantes seront éventuellement mises à jour afin de tenir compte de nouveaux tests élémentaires ayant reçu une réponse certaine.

Exemple 3.4.2 *Supposons que l'on dispose de n mesures définissant n intervalles $[y_s(t_i)]$ et qu'il faille trouver l'ensembles des paramètres p d'un modèle $y_m(t_i, \mathsf{p})$ satisfaisant (3.40). Pour tout $i = 1, \ldots, n$, il est possible de définir le test élémentaire*

$$t_i(\mathsf{p}) = 1 \Leftrightarrow y_m(t_i, \mathsf{p}) \in [y_s(t_i)]. \tag{3.54}$$

Le test global sera alors

$$t(\mathsf{p}) = \bigwedge_{i=1}^{n} t_i(\mathsf{p}), \tag{3.55}$$

et un vecteur de masque pour ce test global est défini en considérant le résultat de chacun des tests pris indépendamment. ♢

3.4.3 Présence de données aberrantes

Toute mesure qui ne satisfait pas les hypothèses faites sur les erreurs, est dite *aberrante*. De telles données peuvent être dues à un mauvais fonctionnement du dispositif de mesure, à une perturbation exceptionnelle, à une limitation du modèle décrivant le processus, etc. Dans de telles conditions, l'ensemble $\mathcal{S}$ pourra éventuellement être vide.

Afin de se protéger vis-à-vis de données aberrantes, on peut chercher à identifier l'ensemble des vecteurs de paramètres qui sont cohérents non plus avec n données, mais $n - q$ données seulement. La fonction *et-q-relaxé* $\bigoplus_q$ présentée au paragraphe 2.7 permet de résoudre ce problème.

Exemple 3.4.3 *Si on reprend l'exemple précédent, le test global $t(\mathsf{p})$ peut être remplacé par*

$$t'(\mathsf{p}) = \bigoplus_{i=1}^{n}{}_q\left(t_i(\mathsf{p})\right), \tag{3.56}$$

qui sera vrai lorsque $n-q$ tests élémentaires seront vrais, c'est-à-dire lorsque $n-q$ sorties du modèle seront compatibles avec les mesures. ♢

Le choix du nombre de données aberrantes q à considérer est un problème délicat. Une solution est apportée par l'algorithme Gomne (*Guaranteed outlier minimal number estimator*, estimateur garanti du nombre minimal de données aberrantes) présenté dans [Jaulin et al.98] et qui constitue une version garantie de Omne, introduit dans [Lahanier et al.87]. Cet algorithme cherche le nombre minimal de données aberrantes $q_{\min}$ tel que l'ensemble $\mathcal{S}_{q_{\min}}$ des paramètres compatibles avec $n-q_{\min}$ données au moins soit non vide. On pourra également choisir de se protéger vis-à-vis d'un nombre $q > q_{\min}$ de données aberrantes, au prix de l'augmentation du volume de l'ensemble $\mathcal{S}_q$.

3.4.4 MaskSivia, Siviautilisant un masque

Les tests d'inclusion utilisant un vecteur de masque sont très faciles à intégrer dans Sivia à l'aide de la fonction récursive RecMSiv (*recursive masked* Sivia, Sivia récursif et utilisant des masques, voir le tableau 3.3). Cette fonction permet de stocker les pavés dans $\mathcal{S}^-$ ou dans $\mathcal{S}^0$ suivant le résultat du test d'inclusion $t_{[]}([\mathsf{p}], [\Pi])$, avec $[\Pi]$ un masque attaché au pavé père de $[\mathsf{p}]$. Afin de stocker des pavés aussi grands que possible dans $\mathcal{S}^-$ et dans $\mathcal{S}^0$, lorsque les deux pavés résultant de la bissection d'un pavé père doivent être stockés tous les deux dans $\mathcal{S}^-$ ou dans $\mathcal{S}^0$, ces deux pavés sont rassemblés en un pavé père unique. Ce processus est itéré aussi longtemps que possible jusqu'à ce que deux pavés frères n'appartiennent plus tous les deux à $\mathcal{S}^0$ ou tous les deux à $\mathcal{S}^-$. L'union des deux masques est renvoyée, car le pavé père doit obtenir les deux informations sur ses pavés fils

RecMSiv est appelé pour la première fois par MaskSivia décrit dans le tableau 3.4. Lorsque la valeur $[t_0]$ renvoyée par RecMSiv à MaskSivia est différente de 0, alors le pavé initial de recherche $[\mathsf{p}_0]$ doit, suivant la valeur de $[t_0]$, être entièrement inclus dans $\mathcal{S}^0$ ou dans $\mathcal{S}^-$. Dans tous les autres cas, $\mathcal{S}^0$ et $\mathcal{S}^-$ ont été construits récursivement par RecMSiv. Le masque est stocké dans l'ensemble solution en même temps que le pavé auquel il est associé. Dans le cas général, cette information n'est pas d'un très grand intérêt, par contre, lorsque des données aberrantes sont présentes, le masque peut aider à leur détection.

Remarque 3.4.2 *Si l'usage de masques permet d'accélérer le comportement des algorithmes en réduisant le nombre de tests à effectuer, ceci se fait au détriment du nombre d'informations à conserver en mémoire dans les vecteurs de masque. Il faut donc rechercher un compromis entre réduction du nombre de tests à effectuer et complexité de manipulation et de stockage des vecteurs de masque.* ♢

3.4.5 Cas où les bornes d'erreurs sont inconnues

Les bornes de l'erreur tolérée sur chaque mesure n'ont pas toujours de réelle justification physique, et il est parfois impossible de les obtenir *a priori*. L'incertitude sur les données est alors un paramètre supplémentaire à identifier. Il convient tout d'abord de choisir une structure pour les bornes sur les

RecMSiv

Entrées : $[\mathbf{p}], [\Pi], \mathcal{S}^-, \mathcal{S}^0, \epsilon$;
Sorties : $[t], [\Pi], \mathcal{S}^-, \mathcal{S}^0$;
// traitement de base
$[t] = t_{\prod}([\mathbf{p}], [\Pi])$;
if $([t] \neq [0,1])$
return $([t], [\Pi], \mathcal{S}^-, \mathcal{S}^0)$;
if $(w([\mathbf{p}]) < \epsilon)$
return $([0,1], [\Pi], \mathcal{S}^-, \mathcal{S}^0)$;
// traitement récursif
bissect($[\mathbf{p}]$,$L[\mathbf{p}]$,$R[\mathbf{p}]$);
$[\Pi]_L = [\Pi]$; $[\Pi]_R = [\Pi]$;
$([t_L], [\Pi]_L, \mathcal{S}^-, \mathcal{S}^0) =$ RecMSiv $(L[\mathbf{p}], [\Pi]_L, \mathcal{S}^-, \mathcal{S}^0, \epsilon)$;
$([t_R], [\Pi]_R, \mathcal{S}^-, \mathcal{S}^0) =$ RecMSiv $(R[\mathbf{p}], [\Pi]_R, \mathcal{S}^-, \mathcal{S}^0, \epsilon)$;
// regroupement si tests identiques sur deux pavés frères
if $([t_L] = [t_R])$
return $([t_L], [\Pi]_L \cup [\Pi]_R, \mathcal{S}^-, \mathcal{S}^0)$;
// stockage si résultats sont différents
if $([t_L] = 1)$
$\mathcal{S}^- = \mathcal{S}^- + (L[\mathbf{p}], [\Pi]_L)$;
if $([t_L] = [0,1])$
$\mathcal{S}^0 = \mathcal{S}^0 + (L[\mathbf{p}], [\Pi]_L)$;
if $([t_R] = 1)$
$\mathcal{S}^- = \mathcal{S}^- + (R[\mathbf{p}], [\Pi]_R)$;
if $([t_R] = [0,1])$
$\mathcal{S}^0 = \mathcal{S}^0 + (R[\mathbf{p}], [\Pi]_R)$;
return $(0, [\Pi]_L \cup [\Pi]_R, \mathcal{S}^-, \mathcal{S}^0)$.

Tableau 3.3: Fonction récursive appelée par MaskSivia.

MaskSivia

Entrées : $[\mathbf{p}_0], \epsilon$;
Sorties : $\mathcal{S}^-, \mathcal{S}^0$;
$\mathcal{S}^- = \emptyset$; $\mathcal{S}^0 = \emptyset$; $[\Pi_0] = [0,1]^n$;
$([t_0], [\Pi], \mathcal{S}^-, \mathcal{S}^0) =$ RecMSiv $([\mathbf{p}_0], [\Pi_0], \mathcal{S}^-, \mathcal{S}^0, \epsilon)$;
if $([t_0] = 1)$
$\mathcal{S}^- = \{[\mathbf{p}_0], [\Pi]\}$;
if $([t_0] = [0,1])$
$\mathcal{S}^0 = \{[\mathbf{p}_0], [\Pi]\}$;
return $(\mathcal{S}^-, \mathcal{S}^0)$.

Tableau 3.4: Algorithme MaskSivia.

erreurs. On pourra par exemple choisir une borne constante e pour l'ensemble des mesures ; $[y_s(t_i)]$ sera alors construit en utilisant la valeur mesurée $y_s(t_i)$ et e comme

$$[y_s(t_i)] = [y_s(t_i) - e, y_s(t_i) + e]. \tag{3.57}$$

Il est également possible de choisir une borne constante e sur l'erreur relative, telle que

$$[y_s(t_i)] = [y_s(t_i)(1-e), y_s(t_i)(1+e)]. \tag{3.58}$$

Remarque 3.4.3 *D'autres structures d'erreurs pourraient être considérées, nous nous limiterons ici au cas où un seul hyperparamètre est utilisé pour représenter l'erreur tolérée sur les mesures.* ♢

Le processus d'identification des paramètres devra donc permettre de déterminer à la fois l'ensemble des valeurs des paramètres et l'hyperparamètre e. Pour réaliser cette tâche, nous proposons l'algorithme Meboe (*Minimal error bound estimator*, estimateur de plus petite borne d'erreur).

Meboe

Entrées : $[e]$, Δe, $[\mathsf{p}_0]$, f, y^s, ϵ_0, $\epsilon_{\min}$
Sorties : e^*, $\mathcal{S}^-$, $\mathcal{S}^0$
Initialisation : $e_{\min} = \underline{e}$, $e_{\max} = \overline{e}$
Repéter
  $e = \dfrac{e_{\max} + e_{\min}}{2}$; $\epsilon = \epsilon_0$; $[y_s(t_i)] = [y_s(t_i) - e, y_s(t_i) + e]$ pour $i = 1, \ldots, n$
  Répéter
    $\left(\mathcal{S}^-, \mathcal{S}^0\right) =$Sivia$([\mathsf{p}_0], \mathsf{f}, [\mathsf{y}^s], \epsilon)$
    $\epsilon = \epsilon/2$
  Jusqu'à ce que $(\mathcal{S}^- \neq \emptyset)$ ou $\left(\mathcal{S}^0 = \emptyset\right)$ ou $(\epsilon < \epsilon_{\min})$
  si $(\mathcal{S}^- \neq \emptyset)$ ou $(\epsilon < \epsilon_{\min})$, $e_{\max} = e$
  sinon si $\mathcal{S}^- = \emptyset$, $e_{\min} = e$
Jusqu'à ce que $e_{\max} - e_{\min} < \Delta e$

Tableau 3.5: Estimateur de la plus petite borne d'erreur.

En entrée, on fournit un intervalle de recherche $[e]$ pour la borne d'erreur ainsi qu'une précision Δe à atteindre pour la caractérisation de e. Soient $[\mathsf{p}_0]$ le pavé de recherche des paramètres, f la fonction à inverser, y^s le vecteur de mesures et ϵ_0 le paramètre de précision initial pour Sivia, $\epsilon_{\min}$, la valeur minimale que peut atteindre ce paramètre. Meboe réalise une recherche de e par dichotomie. A l'aide de la valeur courante de e, le pavé dans lequel les mesures peuvent se trouver est construit. On recherche avec Sivia s'il existe des valeurs des paramètres satisfaisant

$$\mathsf{f}(\mathsf{p}) \in [\mathsf{y}^s], \tag{3.59}$$

avec une précision de découpage allant jusqu'à ϵ. Si $\mathcal{S}^0$ et $\mathcal{S}^-$ sont vides, il ne peut y avoir de valeur de p satisfaisant (3.59), la valeur courante de e est donc trop petite. Si $\mathcal{S}^-$ n'est pas vide, il existe alors des valeurs de p satisfaisant (3.59) ; comme la valeur minimale donnant un ensemble $\mathcal{S}^-$ non vide est

recherchée, à l'itération suivante, il faut diminuer e. Enfin, si $\mathcal{S}^-$ est vide et $\mathcal{S}^0$ ne l'est pas, les pavés incertains stockés dans $\mathcal{S}^0$ peuvent contenir des valeurs de p satisfaisant (3.59). Si le facteur de précision ϵ est supérieur à $2\epsilon_{\mathsf{min}}$, Sivia est à nouveau appliqué avec pour facteur de précision $\epsilon/2$. Dans le cas contraire, un traitement analogue à celui du cas $\mathcal{S}^-$ non vide est effectué. La dichotomie sur e est arrêtée lorsqu'un encadrement de longueur inférieure à Δe est obtenu pour e.

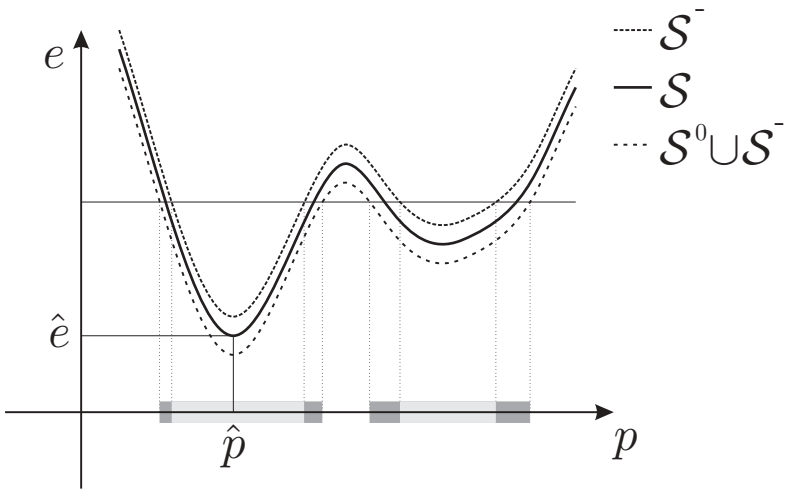


Figure 3-15: Meboe dans le cas unidimentionnel. $\hat{e}$ est la plus petite borne d'erreur permettant d'avoir un ensemble de paramètres admissible $\mathcal{S}$ non vide.

La figure 3-15 illustre le principe de l'algorithme Meboe dans le cas de la recherche d'un seul paramètre p. Pour une précision de recherche ϵ donnée, les ensembles $\mathcal{S}^-$ et $\mathcal{S}^- \cup \mathcal{S}^0$ encadrant $\mathcal{S}$ sont tracés en fonction de la borne sur l'erreur tolérée e. Il s'agit de trouver $\hat{e}$ la plus petite valeur de e telle que l'ensemble $\mathcal{S}^-$ (ou éventuellement $\mathcal{S}^- \cup \mathcal{S}^0$) soit non vide.

Remarque 3.4.4 *Cette version de* Meboe *est très rudimentaire. Comme on recherche la plus petite borne sur l'erreur telle qu'il existe une valeur admissible des paramètres, lors de la dichotomie, il suffit de disposer d'une seule valeur de* p *admissible et de diminuer e tant que cette valeur reste admissible. Dès qu'elle ne l'est plus, il faut trouver une nouvelle valeur de* p *admissible, ou augmenter à nouveau e. En combinant des méthodes ponctuelles et des méthodes ensemblistes, il devrait être possible d'accélérer grandement la recherche de* $\widehat{e}$ *et de* $\widehat{\mathsf{p}}$. ♢

3.5 Identification dans le contexte erreurs bornées

Nous allons considérer un problème traité dans le cadre d'une collaboration avec le *Service Mesure* de Supélec [BB et al.99]. Il s'agit d'examiner la surface intérieure de tubes ferromagnétiques afin de localiser des défauts du type rainures annulaire. La méthode employée repose sur la mesure de courants de Foucault induits dans le tube.

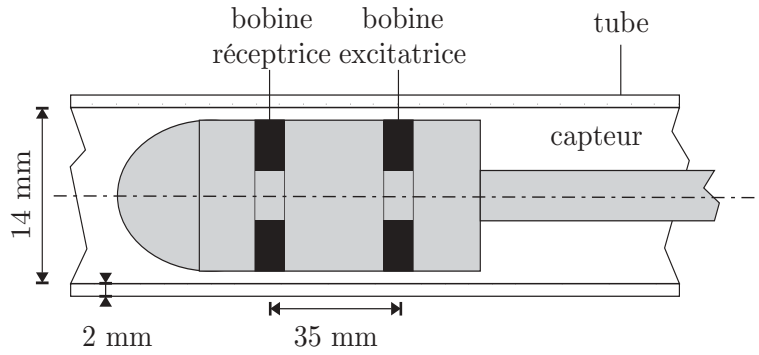


Figure 3-16: Sonde de détection de courants de Foucault induits (échelles non respectées).

La bobine excitatrice (voir la figure 3-16) induit un courant de Foucault dans la paroi du tube ; le champ magnétique créé par le courant de Foucault induit une tension aux bornes de la bobine réceptrice. Les deux bobines sont suffisamment éloignées pour être dans la condition des champs lointains (cf. [Mackintosh et al.96]). Un dispositif mesure le gain et le déphasage entre la tension appliquée à la bobine excitatrice et la tension aux bornes de la bobine réceptrice, et cela pour différentes positions x_i du capteur tout au long du tube. Un défaut dans la structure du tuyau modifie la tension induite. Il s'agit, à l'aide de la mesure des caratéristiques de la tension induite en n points du tuyau à vérifier, de déterminer la longueur l d'une éventuelle rainure ainsi que sa profondeur d, la position du centre de la rainure étant supposée connue (voir la figure 3-17).

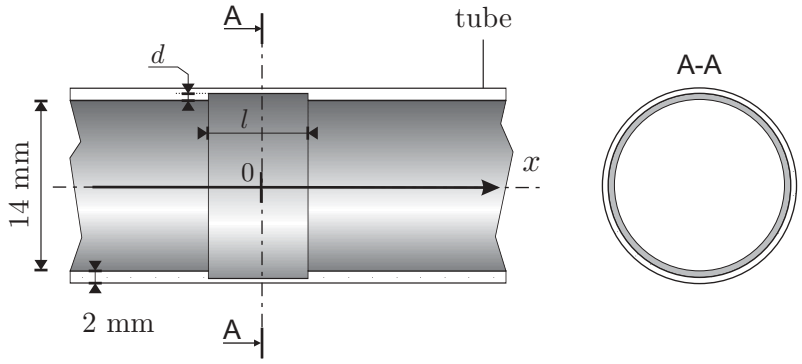


Figure 3-17: Caractéristiques d et l d'un défaut du tube.

Une série-type de mesures de déphasage $y_s(x_i)$ est représentée sur la figure 3-18. Un tube est inspecté sur une longueur de 156 mm ; une mesure est effectuée tous les 1.33 mm, au total on dispose de $n = 118$ points de mesure. Ces données ont été obtenues par simulation en utilisant une méthode par éléments finis [Davoust et al.98] ; la simulation donne des résultats très proches de mesures réelles, qui sont très peu bruitées.

On cherche un modèle comportemental permettant de décrire au mieux l'ensemble de ces données.

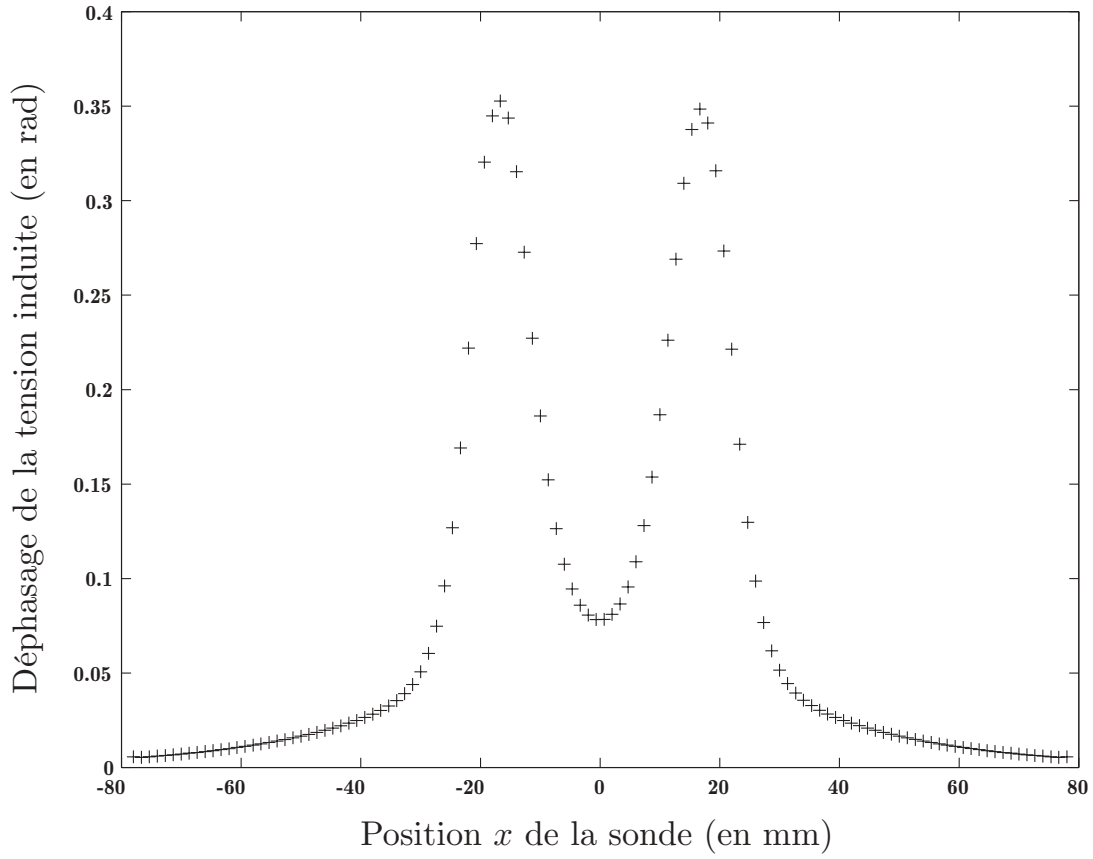


Figure 3-18: Courbe du déphasage mesuré en fonction de la position de la sonde par rapport au centre de la rainure.

Plusieurs structures de modèles ont été suggérées, nous nous bornerons à illustrer les résultats obtenus avec la structure de modèle la plus simple, utilisant un vecteur p à deux paramètres

$$y_{\mathrm{m}}\left(x_i, \mathsf{p}\right) = p_1 \left(\frac{1}{p_2 + \left(x_i - \frac{x_0}{2}\right)^2} + \frac{1}{p_2 + \left(x_i + \frac{x_0}{2}\right)^2} \right). \tag{3.60}$$

Les valeurs de p permettent, après traitement, d'accéder aux caractéristiques l et d du défaut [BB et al.99].

Dans tous les cas traités, nous considérons une structure de bruit sur les mesures de type absolu (3.57). Le coefficient $x_0 = 35$ mm correspond à la distance entre les deux bobines du capteur. L'espace de recherche pour p sera $[0, 100]^2$ dans chacun des cas considés.

3.5.1 Bruit donné **a priori**

Lorsque les bornes du bruit sont connues *a priori*, nous cherchons à caractériser l'ensemble des valeurs des paramètres tel que la sortie générée par le modèle reste compatible avec les mesures.

Pour $e = 0.05$ et $\epsilon = 0.1$, nous obtenons l'ensemble $\mathcal{S}^- \cup \mathcal{S}^0$ représenté sur la figure 3-19. L'ensemble des paramètres admissibles $\mathcal{S}^-$ est non-vide.

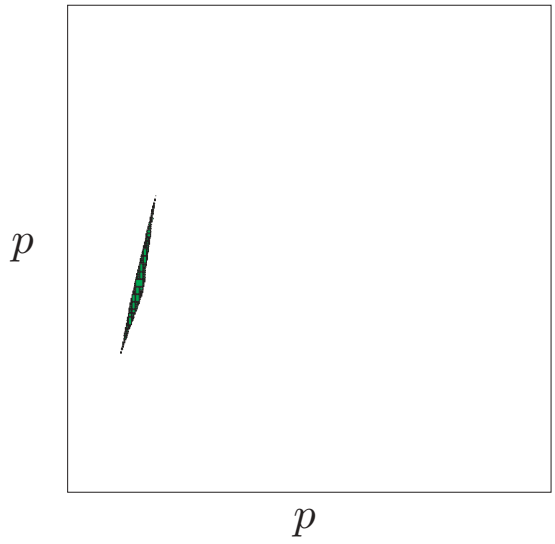


Figure 3-19: Ensemble des p admissibles et incertains avec $\epsilon = 0.05$.

3.5.2 Bruit inconnu

L'hypothèse d'une connaissance *a priori* de e n'a pas beaucoup de sens ici, car les mesures doivent être approximées par un modèle comportemental. Dans ces conditions, on peut chercher la plus petite valeur $\hat{e}$ de la borne d'erreur qui permet d'obtenir un ensemble $\mathcal{S}^-$ non vide. Nous utilisons Meboe pour

caractériser $\hat{e}$ et l'ensemble estimé. La recherche de $\hat{e}$ se fera dans l'intervalle $[0.001, 0.05]$, la précision sur $\hat{e}$ sera $\Delta e = 0.001$ et $\epsilon_{\mathsf{min}} = 0.001$.

En utilisant Meboe, on trouve $\hat{e} \in [0.035, 0.036]$ et l'ensemble solution correspondant est représenté sur la figure 3-20 en moins d'une minute.

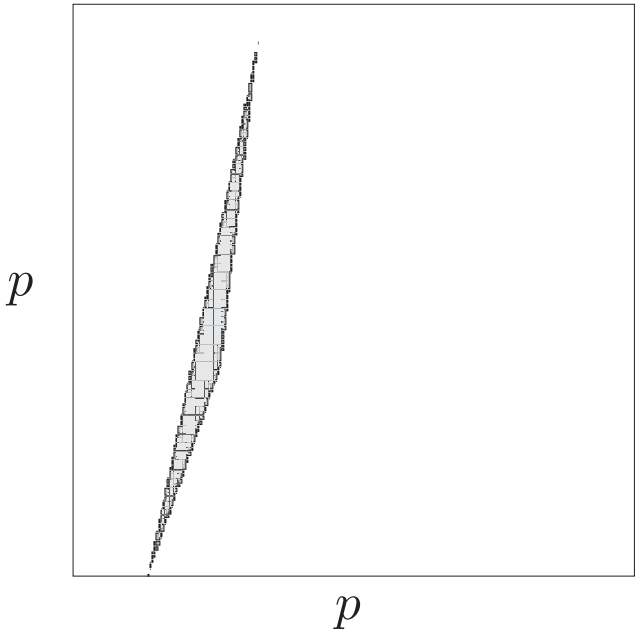


Figure 3-20: Ensemble des p admissibles et incertains avec une borne $e = 0.035$ obtenue par Meboe.

L'ensemble $\mathcal{S}^-$ est représenté en gris clair, l'ensemble $\mathcal{S}^0$ en gris foncé. La figure a été agrandie, et les ensembles obtenus sont inclus dans l'ensemble représenté sur la figure 3-19.

3.6 Conclusion

Ce chapitre a décrit l'apport de l'analyse par intervalles dans le contexte de l'estimation de paramètres. Dans le cadre de l'identification par minimisation d'un critère, une version simplifiée de l'algorithme de Hansen a été présentée. Cet algorithme permet d'obtenir un encadrement de tous les arguments de l'optimum global d'une fonction. Il a été appliqué sur un problème d'estimation des paramètres d'un modèle compartimental et a montré sa capacité à mettre en évidence des défauts d'identifiabilité sans faire d'étude théorique préalable.

Dans le cadre d'erreurs bornées sur les mesures, nous avons présenté une version améliorée de Sivia permettant de manipuler des tests booléens. Les masques ont été introduits afin d'accélérer le traitement des données. Une version robuste de l'estimation de paramètres dans le cadre d'erreurs bornées a été présenté. Nous avons en outre introduit le nouvel -algorithme Meboe qui permet d'obtenir la plus petite borne sur les erreurs qui fournit un ensemble de vecteurs de paramètres non vide. Les propriétés de

convergence et le point de rupture des estimateurs résultants restent à étudier.

La principale limitation des techniques présentées est leur complexité croissante avec le nombre de paramètres du problème et le pessimisme introduit par la présence d'occurences multiples de certaines variables dans les expressions à manipuler. Malgré tout, de nombreux problèmes d'un intérêt pratique réel peuvent être abordés par l'analyse par intervalles, et nous en verrons un exemple concret au chapitre 4.

Lorsque des résultats peuvent être obtenus par l'analyse par intervalles, leur caractère garanti permet de justifier l'augmentation du temps de calcul nécessaire par rapport à des techniques classiques qui n'offrent aucune garantie sur leurs résultats.

Chapitre 4

Localisation statique d'un robot

4.1 Introduction

Les robots sont des dispositifs mécaniques destinés à réaliser des tâches répétitives, peu gratifiantes, dangereuses, ou tout simplement qui nécessitent une force ou une précision que des hommes ne peuvent atteindre. Les premiers modèles n'étaient que des *robots manipulateurs* ; constitués d'un bras disposant d'un certain nombre de degrés de liberté, et solidaires d'un socle fixe, ils ne pouvaient réaliser que des tâches très simples et répétitives dans un environnement limité (ce sont les robots des premières chaînes de fabrication automatisée). La recherche en robotique s'est ensuite intéressée à l'augmentation de l'autonomie des robots, par exemple, par l'ajoût de capteurs, par l'augmentation de leur mobilité, par la mise en œuvre de dispositifs de décision leur permettant de s'adapter de manière autonome au contexte, *etc.* Ainsi, les *robots mobiles* peuvent prendre des formes très diverses selon la tâche qui leur sera dévolue et l'environnement dans lequel ils devront évoluer. Une condition nécessaire à l'autonomie de tels robots est leur capacité à estimer leur état à partir d'informations disponibles *a priori* et de mesures qu'ils sont susceptibles d'effectuer.

Ce chapitre concerne la localisation autonome d'un robot semblable à celui représenté sur la figure 4-1. Cette localisation se fera à partir des informations recueillies par la ceinture de capteurs extéroceptifs, c'est-à-dire sensibles à des informations prises à l'extérieur du robot, visibles sur la bande grise qui entoure le chassis. Dans le cas considéré (robot destiné à d'éventuelles applications grand public), des capteurs à ultrasons, connus pour être très bon marché mais imprécis, sont employés. D'autres types de capteurs, par exemple des télémètres laser, pourraient également être utilisés (pour des applications plus sensibles, par exemple des robots dans des centrales nucléaires) ; la méthodologie développée dans ce chapitre resterait identique. L'environnement dans lequel évolue le robot sera supposé bidimensionnel (bien qu'une extension au cas tridimensionnel ne pose pas de difficultés de principe), et une carte de l'environnement est à disposition du robot. Aucune balise n'est supposée présente dans l'environnement du robot.

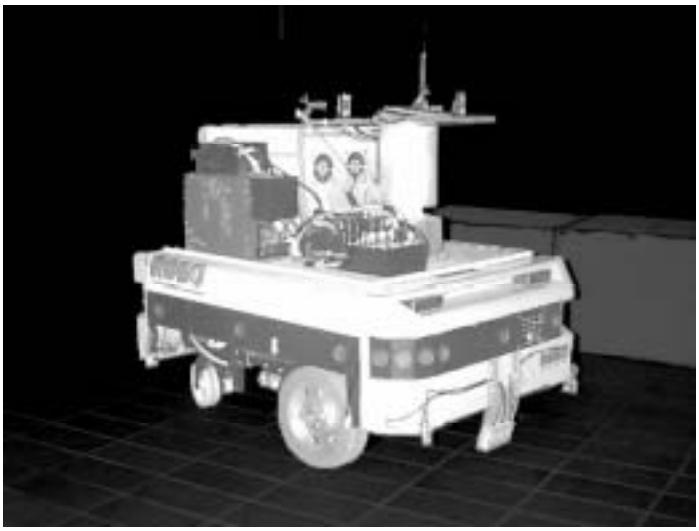


Figure 4-1: *Robuter* : robot mobile de la société *Robosoft*.

Dans ce contexte, les méthodes habituelles de localisation présentent certaines limitations. Dans un premier temps, chaque mesure fournie par un capteur extéroceptif doit être associée à un élément de la carte (voir par exemple [Drumheller87], [Grimson et al.87] ou [Leonard et al.91], qui utilisent un arbre pour explorer les différentes possibilités d'association). Cette étape d'association des mesures à la carte se révèle complexe et gourmande en temps de calcul. Une seconde limitation est que les résultats obtenus ne sont en aucune manière garantis : aucune technique ne garantit que toutes les positions possibles du robot sont obtenues. La dernière limitation de certaines de ces techniques réside en leur faible robustesse par rapport à des données aberrantes, dues par exemple à un dysfonctionnement des capteurs (par exemple [Neira et al.96] ou [Halbwachs et al.97]). Par contre, la méthodologie proposée dans ce chapitre, fondée sur l'estimation de paramètres en faisant l'hypothèse d'erreurs bornées sur les mesures (cf. chapitre 3), n'a pas ces limitations. L'étape d'association des mesures aux données n'est pas nécessaire, le problème est résolu de manière globale. En outre, l'estimateur employé est très robuste vis-à-vis d'éventuelles données aberrantes (voir [Leveque et al.97] et [Kieffer et al.99a]).

Ce chapitre est organisé de la manière suivante. La formulation mathématique du problème est introduite au paragraphe 4.2. Le paragraphe 4.3 est consacré à la description des tests élémentaires qui seront utilisés pour localiser le robot. La combinaison de ces différents tests ainsi que leur extension intervalle sont abordées au paragraphe 4.4. Nous verrons ensuite au paragraphe 4.5 comment, à l'aide de l'algorithme d'inversion ensembliste Sivia, il est possible de caractériser de manière garantie l'ensemble des positions du robot qui sont compatibles avec les informations disponibles. Une attention toute particulière sera accordée à la robustesse de l'estimateur à l'égard de données aberrantes fournies par les capteurs à ultrasons. Enfin, l'algorithme obtenu est appliqué sur trois exemples tests détaillés dans le paragraphe 4.6.

4.2 Formulation du problème

Deux repères sont considérés dans la suite de ce chapitre : le repère $\mathcal{W}$ du laboratoire et un repère $\mathcal{R}$, attaché au robot, ayant pour origine $\mathsf{c} = (x_c, y_c)$ dans $\mathcal{W}$. L'angle entre $\mathcal{R}$ et $\mathcal{W}$ est noté θ ; il correspond à l'orientation du robot (voir la figure 4-2). Les points et leurs coordonnées seront désignés par des lettres minuscules dans $\mathcal{W}$, et surmontées d'un tilde dans $\mathcal{R}$. Ainsi, par exemple, un capteur est désigné par s dans $\mathcal{W}$ et par $\tilde{\mathsf{s}}$ dans $\mathcal{R}$, avec

$$\mathsf{s} = \begin{pmatrix} x_c \\ y_c \end{pmatrix} + \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tilde{\mathsf{s}}. \tag{4.1}$$

Pour décrire la position du robot dans un environnement à deux dimensions, la connaissance de trois paramètres est nécessaire : les coordonnées x_c et y_c de l'origine de $\mathcal{R}$ dans $\mathcal{W}$ et l'orientation θ du robot dans $\mathcal{W}$. Ces trois paramètres forment le vecteur de *configuration* $\mathsf{p} = (x_c, y_c, \theta)^{\mathrm{T}}$ du robot. Lorsqu'on considère un pavé initial de recherche (éventuellement de très grande taille) pour les configurations $[\mathsf{p}_0]$, le problème de la localisation d'un robot peut être formulé comme la caractérisation de l'ensemble

$$\mathcal{S} = \{\mathsf{p} \in [\mathsf{p}_0] \mid t(\mathsf{p}) \text{ est vrai}\}. \tag{4.2}$$

Dans cette équation, $t(\mathsf{p})$ est un test ou une combinaison de tests traduisant le fait que la configuration est cohérente avec les mesures effectuées et les informations *a priori*.

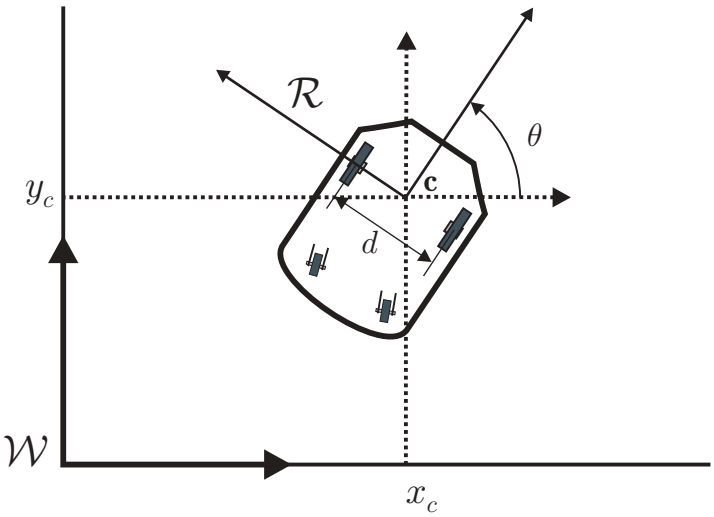


Figure 4-2: Configuration du robot.

4.2.1 Obtention des mesures

Le robot est équipé d'une ceinture de n_{s} capteurs ultrasonores Polaroïd (ce sont en fait des sonars, d'où l'indice s). La position du $i$ème capteur dans le repère $\mathcal{R}$ du robot est $\mathtt{s}_i = (\tilde{x}_i, \tilde{y}_i)$ (dans la suite du chapitre, l'indice i sera plutôt réservé aux capteurs). Ce capteur émet une onde ultrasonore qui est assimilée à un cône (dit d'*émission*) caractérisé par son sommet $\mathtt{s}_i$, son orientation $\tilde{\theta}_i$ et sa demi-ouverture $\tilde{\gamma}_i$ (voir la figure 4-3). Comme la demi-ouverture ne dépend pas du repère considéré, aussi $\tilde{\gamma}_i = \gamma_i$. Ce cône d'émission sera désigné par $\mathcal{C}(\mathtt{s}_i, \tilde{\theta}_i, \tilde{\gamma}_i)$. Le capteur mesure le délai entre les instants d'émission et de réception de l'onde sonore réfléchie ou diffractée par un obstacle quelconque de l'environnement du robot. Ce délai est ensuite converti en une distance d_i. Pour tenir compte des incertitudes de mesure, à chaque distance mesurée d_i est associé un intervalle $[d_i] = [d_i\,(1-\alpha_i)\,, d_i\,(1+\alpha_i)]$, où α_i est l'incertitude relative de mesure caractéristique du capteur i ; cette incertitude sera supposée connue. Par la suite, $[d_i]$ sera appelé *intervalle mesure*. Ainsi, on postule que $[d_i]$ contient la distance du $i$ème capteur à l'obstacle le plus proche se trouvant au moins en partie dans son cône d'émission.

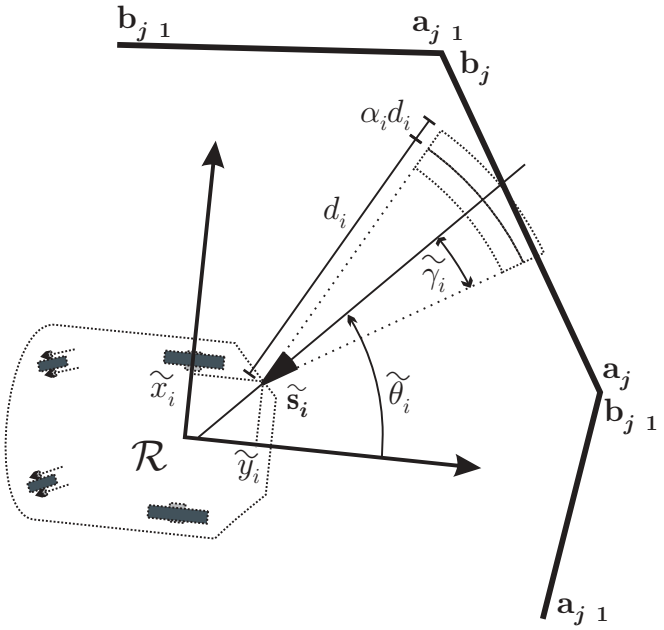


Figure 4-3: Cône d'émission.

4.2.2 Informations **a priori**

Deux types d'informations *a priori* sont considérés. Le premier est une *carte* $\mathcal{M}$ de l'environnement du robot, constituée de n_{w} segments orientés (w pour *walls*) qui décrivent les obstacles (murs, portes, piliers, *etc.*).

$$\mathcal{M} = \{[\mathtt{a}_j, \mathtt{b}_j]\,|j = 1, ..., n_{\mathrm{w}}\}\,. \tag{4.3}$$

Par convention, lors d'un déplacement de a_j vers b_j, la partie réfléchissante du segment est sur la gauche (la matière du mur est à droite). Par la suite, l'indice j sera plutôt réservé aux segments de la carte. Le demi-plan $\Delta_{\mathsf{a_j\,b_j}}$ correspondant au côté réfléchissant du segment $[\mathsf{a}_j, \mathsf{b}_j]$ peut donc être caractérisé par l'équation

$$\Delta_{\mathsf{a_j\,b_j}} = \left\{ \mu \in \mathbf{R}^2 \,\middle|\, \det\left(\overrightarrow{\mathsf{a}_j\mathsf{b}_j}, \overrightarrow{\mathsf{a}_j\mu}\right) \geq 0 \right\}. \tag{4.4}$$

Remarque 4.2.1 *Le produit scalaire et le déterminant sont des outils très commodes pour délimiter des demi-plans ou des secteurs angulaires dans* $\mathbf{R}^2$. *La figure 4-4 présente la définition d'un demi-plan par le test du signe d'un déterminant* (a) *et d'un produit scalaire* (b).

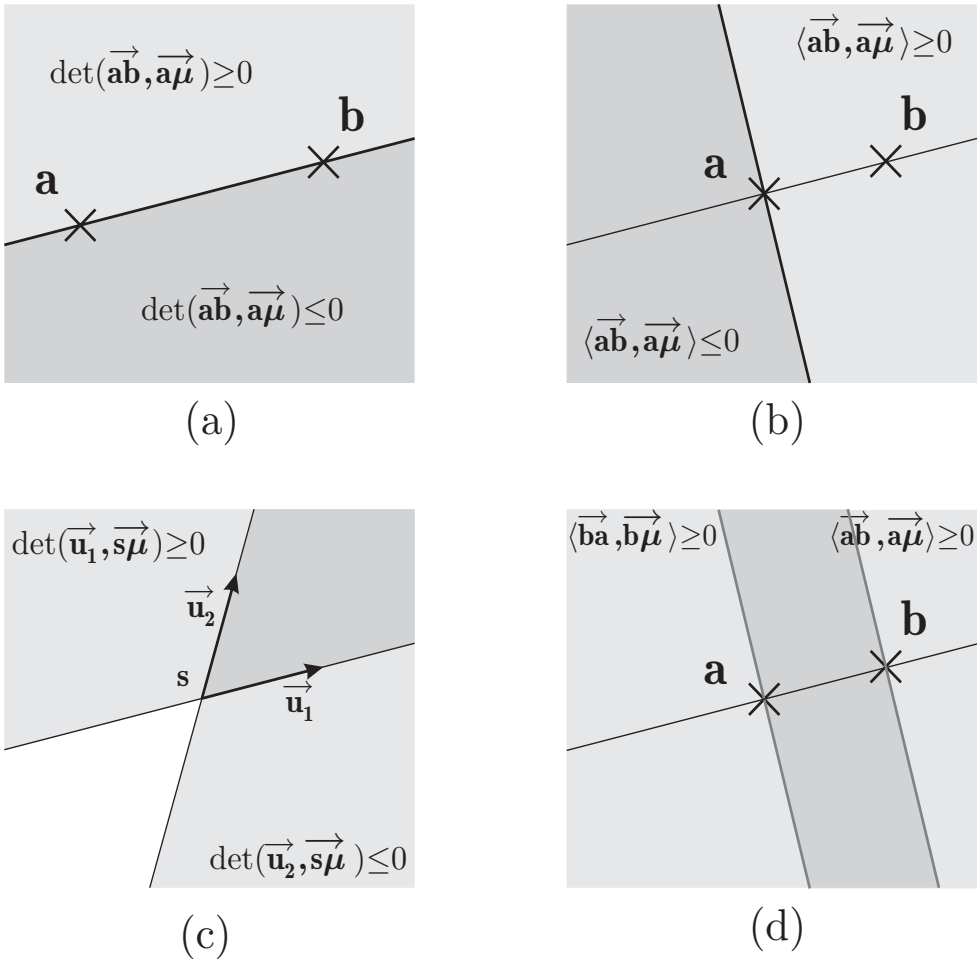


Figure 4-4: Définition de demi-plans, d'un cône et d'un bandeau à l'aide de produits scalaires et de déterminants.

La combinaison de deux déterminants permet de définir ainsi un cône (c), *la combinaison de deux produits scalaires permet de définir un* bandeau (d). ♢

Le second type d'information (optionnelle) est la connaissance d'un ensemble décrit par un polygone dans lequel p se trouve *a priori*. Ce polygone est souvent constitué de tout ou partie de la carte.

4.3 Tests de localisation

Ce paragraphe sera consacré à une énumération de différents tests élémentaires permettant la construction d'un test global $t(\mathsf{p})$ utilisé pour la définition de $\mathcal{S}$ (voir (4.2)). Ces tests seront fondés sur les distances mesurées et sur les informations *a priori*.

4.3.1 Test d'association des données avec la carte

Le test $t(\mathsf{p})$ doit répondre à la question suivante. Si le robot était dans une configuration p, ses capteurs fourniraient-ils des distances identiques (aux incertitudes près) à celles qui ont été effectivement mesurées ? Pour bâtir ce test, les informations disponibles dans le repère du robot $\mathcal{R}$ sont traduites dans le repère du laboratoire $\mathcal{W}$.

Considérons un capteur à ultrasons $\tilde{\mathsf{s}}$, dont le cône d'émission est $\mathcal{C}\left(\tilde{\mathsf{s}}, \tilde{\theta}, \tilde{\gamma}\right)$ (dans ce paragraphe, les indices i et j sont supprimés pour alléger les notations). Pour toute configuration p, $\mathcal{C}$ est décrit de manière équivalente dans $\mathcal{W}$ par son sommet $\mathsf{s}(\mathsf{p})$ et deux vecteurs unitaires $\overrightarrow{\mathsf{u_1}}\left(\mathsf{p}, \tilde{\theta}, \tilde{\gamma}\right)$ et $\overrightarrow{\mathsf{u_2}}\left(\mathsf{p}, \tilde{\theta}, \tilde{\gamma}\right)$ correspondant à ses arêtes. Ainsi, on peut aussi noter le cône $\mathcal{C}\left(\mathsf{s}, \overrightarrow{\mathsf{u_1}}, \overrightarrow{\mathsf{u_2}}\right)$ (en omettant la dépendance en $\mathsf{p}, \tilde{\theta}$ et $\tilde{\gamma}$). Par convention, $\overrightarrow{\mathsf{u_1}}$ et $\overrightarrow{\mathsf{u_2}}$ sont choisis de telle manière que $\overrightarrow{\mathsf{u_2}}$ résulte de $\overrightarrow{\mathsf{u_1}}$ par une rotation d'un angle $2\tilde{\gamma}$ dans le sens trigonométrique. Comme $\tilde{\gamma}$ est physiquement toujours inférieur à $\pi/2$, la condition d'appartenance d'un point quelconque $\mu \in \mathbf{R}^2$ au cône d'émission se traduit par (voir la figure 4-4(c))

$$\mu \in \mathcal{C}\left(\mathsf{s}, \overrightarrow{\mathsf{u_1}}, \overrightarrow{\mathsf{u_2}}\right) \Leftrightarrow \left(\det\left(\overrightarrow{\mathsf{u_1}}, \overrightarrow{\mathsf{s}\mu}\right) \geq 0\right) \wedge \left(\det\left(\overrightarrow{\mathsf{u_2}}, \overrightarrow{\mathsf{s}\mu}\right) \leq 0\right). \tag{4.5}$$

L'algorithme testant la cohérence d'une configuration avec les données est fondé sur la notion d'*espacement* (en anglais *remoteness*) entre un capteur et la carte. Nous allons dans un premier temps définir ce concept pour un capteur et un segment $[\mathsf{a}, \mathsf{b}]$ de la carte pris isolément.

Définition 4.3.1 *L'espacement* $r\left(\mathsf{s}, \overrightarrow{\mathsf{u_1}}, \overrightarrow{\mathsf{u_2}}, \mathsf{a}, \mathsf{b}\right)$ *entre le cône* $\mathcal{C}\left(\mathsf{s}, \overrightarrow{\mathsf{u_1}}, \overrightarrow{\mathsf{u_2}}\right)$ *et le segment* $[\mathsf{a}, \mathsf{b}]$ *est la plus petite distance entre* s *et l'intersection (lorsqu'elle existe) de* $[\mathsf{a}, \mathsf{b}]$ *avec le cône* $\mathcal{C}$*. Le point* s *doit en outre se trouver dans le demi-plan correspondant à la partie réfléchissante de* $[\mathsf{a}, \mathsf{b}]$*. L'espacement est évalué comme suit*

$$\begin{aligned} r\left(\mathsf{s}, \overrightarrow{\mathsf{u_1}}, \overrightarrow{\mathsf{u_2}}, \mathsf{a}, \mathsf{b}\right) &= \infty \text{ si } \mathsf{s} \notin \Delta_{\mathsf{ab}} \text{ ou si } [\mathsf{a}, \mathsf{b}] \cap \mathcal{C} = \emptyset, \\ &= \min_{\mu \in [\mathsf{a}, \mathsf{b}] \cap \mathcal{C}} \|\overrightarrow{\mathsf{s}\mu}\| \text{ dans les autres cas.} \end{aligned} \tag{4.6}$$

♢

En pratique, cet espacement (4.6) est calculé comme suit. L'équation (4.4) est tout d'abord utilisée

pour déterminer si $\mathsf{s} \in \Delta_{\mathsf{ab}}$, c'est-à-dire si s appartient au côté réfléchissant du segment $[\mathsf{a}, \mathsf{b}]$. La figure 4-5(a) présente une situation où $\mathsf{s} \notin \Delta_{\mathsf{ab}}$.

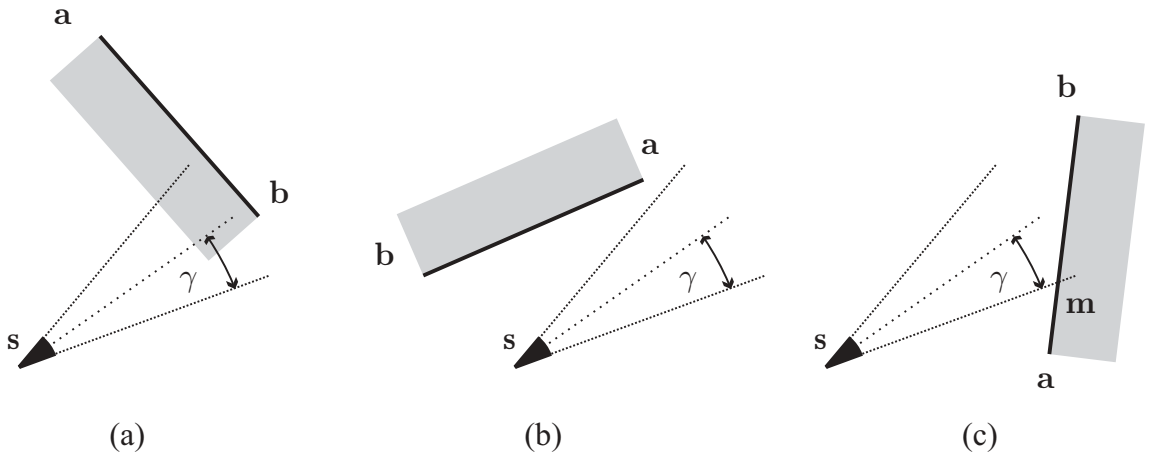


Figure 4-5: Différentes positions d'un cône par rapport à un segment.

Lorsque $\mathsf{s} \in \Delta_{\mathsf{ab}}$, on cherche le minimum de $\|\overrightarrow{\mathsf{s\mu}}\|$ lorsque μ décrit $[\mathsf{a}, \mathsf{b}] \cap \mathcal{C}$. Cette minimisation impose l'examen de plusieurs situations. Soit h la projection orthogonale de s sur la droite (a, b). Si $\mathsf{h} \in [\mathsf{a}, \mathsf{b}] \cap \mathcal{C}$, alors $r\left(\mathsf{s}, \overrightarrow{u_1}, \overrightarrow{u_2}, \mathsf{a}, \mathsf{b}\right) = \left\|\overrightarrow{\mathsf{sh}}\right\|$. Pour tester si $\mathsf{h} \in [\mathsf{a}, \mathsf{b}] \cap \mathcal{C}$, il faut montrer dans un premier temps que $\mathsf{h} \in [\mathsf{a}, \mathsf{b}]$. Comme h est la projection orthogonale de s sur $[\mathsf{a}, \mathsf{b}]$, il faut que s appartienne au bandeau illustré sur la figure 4-4(d), par conséquent

$$\begin{aligned} \mathsf{h} \in [\mathsf{a}, \mathsf{b}] &\iff \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{\mathsf{as}} \right\rangle \geq 0\right) \wedge \left(\left\langle \overrightarrow{\mathsf{ba}}, \overrightarrow{\mathsf{bs}} \right\rangle \geq 0\right) \\ &\iff \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{\mathsf{sa}} \right\rangle \geq 0\right) \wedge \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{\mathsf{sb}} \right\rangle \geq 0\right) \end{aligned} \tag{4.7}$$

De plus, il faut que $\mathsf{h} \in \mathcal{C}$. En utilisant (4.5), on obtient

$$\mathsf{h} \in \mathcal{C} \iff \left(\det\left(\overrightarrow{u_1}, \overrightarrow{\mathsf{sh}}\right) \geq 0\right) \wedge \left(\det\left(\overrightarrow{u_2}, \overrightarrow{\mathsf{sh}}\right) \leq 0\right), \tag{4.8}$$

cette condition n'est cependant pas exploitable, car il faut connaître les coordonnées de h, qui sont relativement complexes à obtenir. Or, si s est du coté réfléchissant du segment $[\mathsf{a}, \mathsf{b}]$, l'angle entre $\overrightarrow{\mathsf{sh}}$ et $\overrightarrow{\mathsf{ab}}$ est de $\frac{\pi}{2}$. par conséquent,

$$\det\left(\overrightarrow{u_1}, \overrightarrow{\mathsf{sh}}\right) \geq 0 \iff \left\langle \overrightarrow{u_1}, \overrightarrow{\mathsf{ab}} \right\rangle \geq 0 \text{ et } \det\left(\overrightarrow{u_2}, \overrightarrow{\mathsf{sh}}\right) \leq 0 \iff \left\langle \overrightarrow{u_2}, \overrightarrow{\mathsf{ab}} \right\rangle \leq 0. \tag{4.9}$$

ainsi, (4.8) devient après quelques manipulations simples

$$\mathsf{h} \in \mathcal{C} \Longleftrightarrow \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{\mathsf{u_1}} \right\rangle \geq 0\right) \wedge \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{\mathsf{u_2}} \right\rangle \leq 0\right). \tag{4.10}$$

En combinant (4.7) et (4.10), on obtient finalement une condition d'appartenance de h à $[\mathsf{a},\mathsf{b}] \cap \mathcal{C}$ ne dépendant pas des coordonnées de h.

$$\mathsf{h} \in [\mathsf{a},\mathsf{b}] \cap \mathcal{C} \Longleftrightarrow \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{\mathsf{sa}} \right\rangle \leq 0\right) \wedge \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{\mathsf{sb}} \right\rangle \geq 0\right) \wedge \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{\mathsf{u_1}} \right\rangle \leq 0\right) \wedge \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{\mathsf{u_2}} \right\rangle \geq 0\right). \tag{4.11}$$

Si $\mathsf{h} \notin [\mathsf{a},\mathsf{b}] \cap \mathcal{C}$, la distance minimale peut soit être infinie (lorsque $[\mathsf{a},\mathsf{b}] \cap \mathcal{C} = \emptyset$, situation illustrée par la figure 4-5(b), soit être obtenue pour un point μ correspondant à l'une des extrémités du segment $[\mathsf{a},\mathsf{b}] \cap \mathcal{C}$, situation illustrée par la figure 4-5(c).

Dans la suite, nous allons considérer cette situation. Soit $\mathsf{h_1}$ et $\mathsf{h_2}$ les intersections de la droite (a,b) avec les droites $(\mathsf{s},\overrightarrow{\mathsf{u_1}})$ et $(\mathsf{s},\overrightarrow{\mathsf{u_2}})$. L'ensemble des extrémités potentielles de $[\mathsf{a},\mathsf{b}] \cap \mathcal{C}$ est donc $\mathcal{K} = \{\mathsf{a},\mathsf{b},\mathsf{h_1},\mathsf{h_2}\}$. De ce fait, si $\mathsf{h} \notin [\mathsf{a},\mathsf{b}] \cap \mathcal{C}$, $r\left(\mathsf{s},\overrightarrow{\mathsf{u_1}},\overrightarrow{\mathsf{u_2}},\mathsf{a},\mathsf{b}\right)$ est soit infinie, soit égale à $\|\overrightarrow{\mathsf{s\mu}}\|$, avec μ appartenant à $\mathcal{K}$. Dans le cas de la figure 4-6, $r\left(\mathsf{s},\overrightarrow{\mathsf{u_1}},\overrightarrow{\mathsf{u_2}},\mathsf{a},\mathsf{b}\right) = \left\|\overrightarrow{\mathsf{sh_2}}\right\|$.

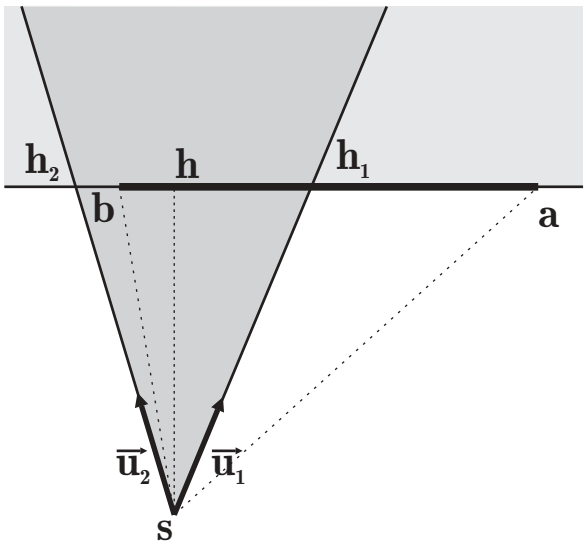


Figure 4-6: Espacement entre un capteur et un segment $[\mathsf{a},\mathsf{b}]$ isolé.

Un test pour déterminer si un élément de $\mathcal{K}$ appartient à l'ensemble $[\mathsf{a},\mathsf{b}] \cap \mathcal{C}$ est facile à obtenir à partir de l'équation (4.5). Soit $\mathsf{v} \in \{\mathsf{a},\mathsf{b}\}$

$$\mathsf{v} \in \mathcal{C} \Longleftrightarrow \left(\det\left(\overrightarrow{\mathsf{u_1}},\overrightarrow{\mathsf{sv}}\right) \geq 0\right) \wedge \left(\det\left(\overrightarrow{\mathsf{u_2}},\overrightarrow{\mathsf{sv}}\right) \leq 0\right). \tag{4.12}$$

Par construction, $\mathsf{h}_i \in \mathcal{C} \cap (\mathsf{a},\mathsf{b})$, il suffit alors de tester si h_i appartient à $[\mathsf{a},\mathsf{b}]$, ce qui revient tester si

$\mathsf{h}_i \in \mathcal{C}\left(\mathsf{s}, \frac{\overrightarrow{\mathsf{sa}}}{\|\overrightarrow{\mathsf{sa}}\|}, \frac{\overrightarrow{\mathsf{sb}}}{\|\overrightarrow{\mathsf{sb}}\|}\right)$. Ainsi, pour $i = 1, 2$,

$$\mathsf{h}_i \in [\mathsf{a}, \mathsf{b}] \cap \mathcal{C} \Longleftrightarrow \left(\det\left(\frac{\overrightarrow{\mathsf{sa}}}{\|\overrightarrow{\mathsf{sa}}\|}, \overrightarrow{\mathsf{sh}_i}\right) \geq 0\right) \wedge \left(\det\left(\frac{\overrightarrow{\mathsf{sb}}}{\|\overrightarrow{\mathsf{sb}}\|}, \overrightarrow{\mathsf{sh}_i}\right) \leq 0\right), \tag{4.13}$$

or comme $\frac{\overrightarrow{\mathsf{sh}_i}}{\|\overrightarrow{\mathsf{sh}_i}\|} = \overrightarrow{u_i}$, on a

$$\mathsf{h}_i \in [\mathsf{a}, \mathsf{b}] \cap \mathcal{C} \Longleftrightarrow \left(\det\left(\overrightarrow{\mathsf{sa}}, \overrightarrow{\mathsf{u}_i}\right) \geq 0\right) \wedge \left(\det\left(\overrightarrow{\mathsf{sb}}, \overrightarrow{\mathsf{u}_i}\right) \leq 0\right). \tag{4.14}$$

En résumé, lorsque ni h, ni aucun élément de $\mathcal{K}$ n'appartient à $[\mathsf{a}, \mathsf{b}] \cap \mathcal{C}$, alors il faut conclure que $[\mathsf{a}, \mathsf{b}] \cap \mathcal{C} = \emptyset$, et l'espacement entre le cône et le segment est infini, dans les autres cas, l'éloignement est fini. L'annexe B.1 présente une fonction construite à partir des tests décrits précédemment et évaluant $r\left(\mathsf{s}, \overrightarrow{\mathsf{u}_1}, \overrightarrow{\mathsf{u}_2}, \mathsf{a}, \mathsf{b}\right)$ pour un segment $[\mathsf{a}, \mathsf{b}]$ isolé.

Remarque 4.3.1 *La version présente du calcul de l'éloignement ne tient pas compte du fait qu'au delà d'un certain angle d'incidence, l'onde réfléchie ou réfractée par un obstacle (si la rugosité de l'obstacle est de l'ordre de la longueur d'onde des ultra-sons) ne peut plus être reçue par le capteur. Il serait assez simple de prendre en compte ce fait en testant les angles d'incidence limites de l'onde émise.* ♢

Dans la situation normale où la carte comprend n_{w} segments, il faut prendre en compte le fait qu'un segment puisse ne pas avoir réfléchi l'onde sonore parce qu'il est masqué par un autre segment, plus proche du capteur à ultrasons. Soit $r_{ij}(\mathsf{p})$ l'espacement entre le $j$ème segment, considéré comme isolé, et le cône associé au $i$ème capteur lorsque la configuration est p ; il est exprimé par

$$r_{ij}\left(\mathsf{p}\right) = r\left(\mathsf{s}_i\left(\mathsf{p}\right), \mathsf{u}_{1i}\left(\mathsf{p}, \tilde{\theta}_i, \tilde{\gamma}_i\right), \mathsf{u}_{2i}\left(\mathsf{p}, \tilde{\theta}_i, \tilde{\gamma}_i\right), \mathsf{a}_j, \mathsf{b}_j\right). \tag{4.15}$$

L'espacement r_i entre le capteur i et tous les segments de la carte (entre le capteur i et la carte, pour abréger) est la plus petite distance entre le capteur et un segment de la carte se trouvant en partie dans le cône d'émission du capteur. Lorsque la configuration est p, $r_i\left(\mathsf{p}\right)$ est donc donné par

$$r_i\left(\mathsf{p}\right) = \min_{j=1,\ldots,n_{\mathrm{w}}} r_{ij}\left(\mathsf{p}\right). \tag{4.16}$$

Exemple 4.3.1 *Considérons la figure 4-7.* $r_{i2}\left(\mathsf{p}\right) = +\infty$ *car* s_i *ne se trouve pas sur la face réfléchissante de* $[\mathsf{a}_2, \mathsf{b}_2]$. $r_i\left(\mathsf{p}\right) = r_{i1}\left(\mathsf{p}\right)$ *car le segment* $[\mathsf{a}_3, \mathsf{b}_3]$ *est masqué par le segment* $[\mathsf{a}_1, \mathsf{b}_1]$, *ce qui se traduit par* $r_{i1}\left(\mathsf{p}\right) \leq r_{i3}\left(\mathsf{p}\right)$. ♢

La mesure d_i fournie par le $i$ème capteur peut être interprétée comme étant due à un obstacle se trouvant dans le cône d'émission du capteur à une distance appartenant à $[d_i]$, compte-tenu de l'incertitude de mesure. La configuration p est cohérente avec la $i$ème mesure si $r_i\left(\mathsf{p}\right)$ appartient à $[d_i]$. D'où le

Test $dat_i\left(\mathsf{p}\right)$: $dat_i\left(\mathsf{p}\right)$ est vrai si et seulement si $r_i\left(\mathsf{p}\right) \in [d_i]$. ♢

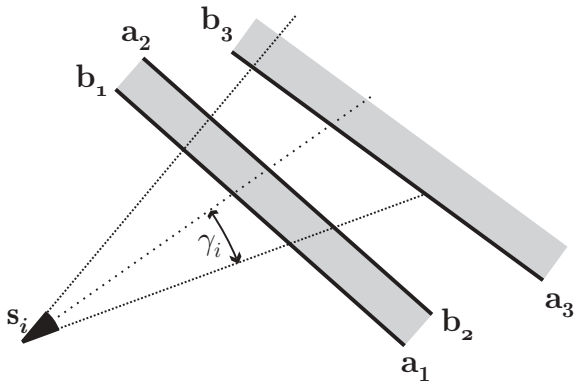


Figure 4-7: Le segment $[\mathsf{a}_3, \mathsf{b}_3]$ est masqué par le segment $[\mathsf{a}_1, \mathsf{b}_1]$.

4.3.2 Test utilisant seulement la carte

Considérons une pièce dans laquelle se trouvent des obstacles tels que des caisses, des piliers, ou du mobiler. Lorsqu'il faut localiser un robot à l'intérieur de cette pièce, on peut immédiatement dire que d'une part, le robot se trouve à l'intérieur de l'espace délimité par les murs de la pièce (c'est l'intérieur de la pièce) et que d'autre part qu'il ne se trouve pas dans l'espace au sol occupé par les meubles, les piliers ou le mobiler. Lorsqu'on dispose d'une carte répertoriant fidèlement chacun de ces objets, il est possible d'obtenir un test n'utilisant que les informations de cette carte pour éliminer certaines configurations.

Pour définir ce test, considérons tout d'abord l'exemple simple illustré par la figure 4-8 : quatre segments orientés délimitent une pièce carrée. L'intérieur de cette pièce (situé du côté réfléchissant des segments) est noté $\mathcal{P}_{\text{int}}$ et l'extérieur $\mathcal{P}_{\text{ext}}$. Pour tester si un point μ est à l'intérieur ou à l'extérieur du carré, il suffit de calculer la somme des angles θ_j entre $\overrightarrow{\mu\mathsf{a}_j}$ et $\overrightarrow{\mu\mathsf{b}_j}$ notés $\theta_j = \arg\left(\overrightarrow{\mu\mathsf{a}_j}, \overrightarrow{\mu\mathsf{b}_j}\right)$. En supposant le plan orienté dans le sens trigonométrique, dans le cas (a) $\sum_{j=1}^{4} \theta_j = 2\pi$, le point μ appartient à $\mathcal{P}_{\text{int}}$; par contre dans le cas (b) $\sum_{j=1}^{4} \theta_j = 0$ et le point μ appartient à $\mathcal{P}_{\text{ext}}$.

Cet exemple permet de comprendre intuitivement la méthode de test des configurations définie dans la suite : on cherche à construire un contour à l'intérieur duquel le robot est supposé appartenir *a priori*. Une démonstration en sera donnée dans l'annexe C.

Supposons qu'il existe un entier p tel qu'il soit possible de reparamétriser les segments de la carte $\mathcal{M}$ de manière à l'écrire

$$\mathcal{M} = \left\{ [\mathsf{a}_{11}, \mathsf{b}_{11}], \ldots, [\mathsf{a}_{1n_1}, \mathsf{b}_{1n_1}], [\mathsf{a}_{21}, \mathsf{b}_{21}], \ldots, [\mathsf{a}_{2n_2}, \mathsf{b}_{2n_2}], \ldots, [\mathsf{a}_{p1}, \mathsf{b}_{p1}], \ldots, [\mathsf{a}_{pn_\mathsf{p}}, \mathsf{b}_{pn_\mathsf{p}}] \right\} \quad (4.17)$$

avec

$$\mathsf{b}_{k1} = \mathsf{a}_{k2}, \ldots, \mathsf{b}_{kl} = \mathsf{a}_{kl+1}, \ldots, \mathsf{b}_{kn_\mathsf{k}} = \mathsf{a}_{k1}, \text{ pour } 1 \leq k \leq p. \quad (4.18)$$

La carte est donc constituée de p polygones fermés indicés par k et composés de n_k segments ($k = 1, \ldots, p$).

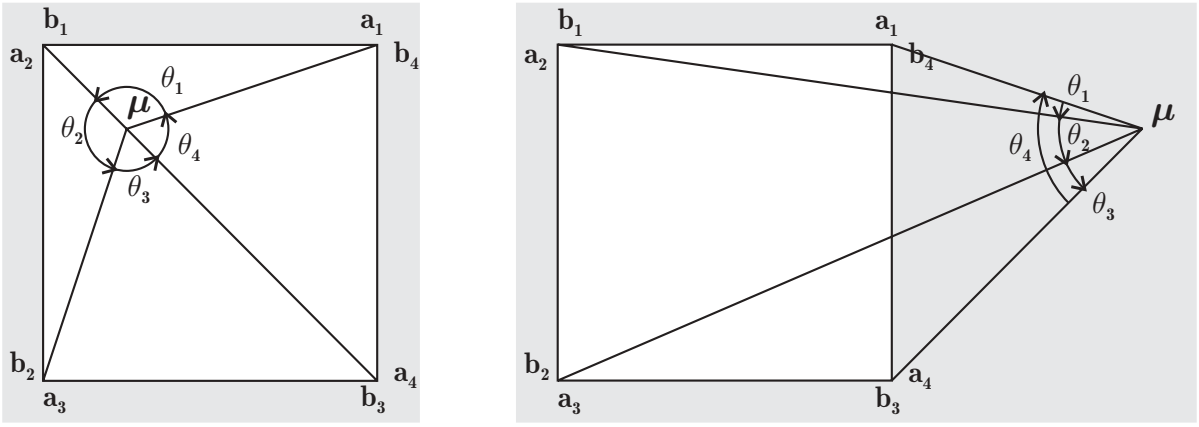


Figure 4-8: μ à l'intérieur (a) et à l'extérieur (b) du carré.

Définissons le contour du $k$ième polygone

$$\mathcal{M}'_k = (\mathsf{a}_{k1}, \mathsf{b}_{k1}, \mathsf{a}_{k2}, \mathsf{b}_{k2}, \ldots, \mathsf{a}_{kn_k}, \mathsf{b}_{kn_k}), \tag{4.19}$$

et le contour global passant par les p polygones par

$$\mathcal{M}' = \left(\mathcal{M}'_1, \mathcal{M}'_2, \mathsf{b}_{1n_1}, \ldots, \mathcal{M}'_k, \mathsf{b}_{1n_1}, \ldots, \mathcal{M}'_p, \mathsf{b}_{1n_1}\right). \tag{4.20}$$

Le polygone $\mathcal{M}'_1$ est particularisé, car son point b_{1n_1} est utilisé pour construire un contour fermé englobant chacun des polygones (voir la figure 4-9, où $p = 4$). Il serait parfaitement possible d'utiliser un ou plusieurs autres polygones pour parvenir à cette fin. Notons encore que $\mathcal{M}'$ fait apparaître des segments *fictifs* (par exemple $[\mathsf{b}_{1n_1}, \mathsf{a}_{21}]$ entre $\mathcal{M}'_1$ et $\mathcal{M}'_2$ ou $[\mathsf{b}_{2n_2}, \mathsf{b}_{1n_1}]$ entre $\mathcal{M}'_2$ et $\mathcal{M}'_1$) ; dans l'annexe C, nous montrons que ceux-ci n'ont aucune influence sur le test défini dans la suite de ce paragraphe.

Proposition 3 *Soient un point* $\mu \in \mathbf{R}^2$ *et une carte* $\mathcal{M}$ *vérifiant* (4.17) *et* (4.18) *et définissant un contour fermé* $\mathcal{M}'$ *tel que* $\mu \notin \mathcal{M}'$.

μ *est à l'intérieur de* $\mathcal{M}'$ *si et seulement si* $\displaystyle\sum_{k=1}^{p}\left(\sum_{l=1}^{n_p}\arg\left(\overrightarrow{\mu\mathsf{a}_{kl}}, \overrightarrow{\mu\mathsf{b}_{kl}}\right)\right) = 2\pi.$

♢

Preuve : voir l'annexe C.

Dans la suite de l'exposé, nous supposerons que les segments de la carte sont déjà ordonnés en une succession de polygones, ainsi, nous continuerons à l'écrire sous la forme (4.3).

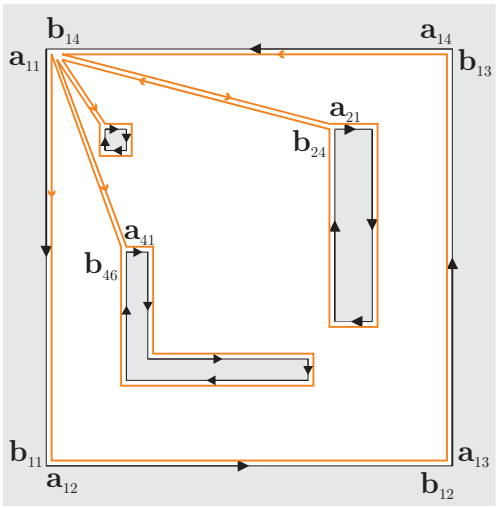


Figure 4-9: Contour fermé défini à partir de la carte et partition de l'espace.

Lorsque la pièce est encombrée d'objets répertoriés sur la carte, pour déterminer si μ se trouve à l'intérieur de l'espace libre (c'est-à-dire non encombré) de la pièce, un test utilisant la proposition 3 peut être utilisé. En effet, si la carte est de la forme (4.17) et si ses segments vérifient (4.18), une partition de l'espace en deux ensembles est réalisée : l'*intérieur*, auquel le robot est supposé appartenir

$$\mathcal{P}_{\text{int}} = \left\{ \mu \in \mathbf{R}^2 \left| \sum_{j=1}^{n_w} \arg\left(\overrightarrow{\mu a_j}, \overrightarrow{\mu b_j} \right) = 2\pi \right. \right\}, \tag{4.21}$$

et l'*extérieur*

$$\mathcal{P}_{\text{ext}} = \left\{ \mu \in \mathbf{R}^2 \left| \sum_{j=1}^{n_w} \arg\left(\overrightarrow{\mu a_j}, \overrightarrow{\mu b_j} \right) = 0 \right. \right\}. \tag{4.22}$$

L'angle entre $\overrightarrow{\mu a_j}$ et $\overrightarrow{\mu b_j}$, est contraint à appartenir à $]-\pi, \pi[$. Cette contrainte n'est évidemment pas appliquée à la somme des arguments afin de distinguer 0 de 2π. La figure 4-9 illustre une division de l'espace à l'aide de tous les segments de la carte. $\mathcal{P}_{\text{int}}$ est en blanc et $\mathcal{P}_{\text{ext}}$ en gris. Sur chaque segment, la flèche indique la direction de parcours du segment, de a vers b. Gardons à l'esprit que la face réfléchissante est toujours sur la gauche du segment lors d'un déplacement de a vers b.

Remarque 4.3.2 *L'hypothèse de l'organisation des segments de la carte en une succession de polygones ((4.17) et (4.18)) peut paraître assez forte. Lorsqu'elle n'est pas réalisée, d'autres segments fictifs non réfléchissants peuvent être ajoutés à la carte pour définir $\mathcal{P}_{int}$ et $\mathcal{P}_{ext}$. Sur la figure 4-10, la pièce où se trouve le robot dispose d'une porte ouverte. $\mathcal{P}_{int}$ et $\mathcal{P}_{ext}$ peuvent malgré tout être définis par l'ajoût à la carte du segment fictif $[a_6, b_6]$ (non réféchissant) mis à la place du pas de la porte.* ♢

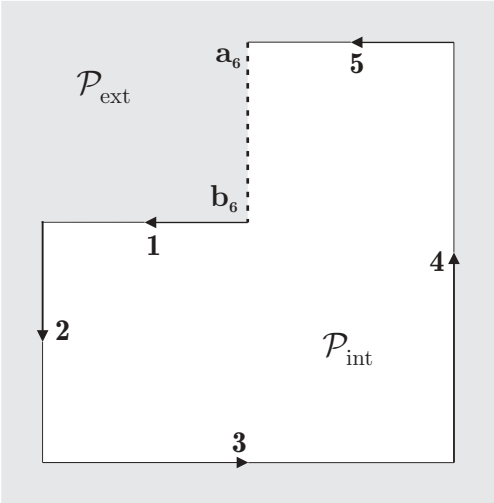


Figure 4-10: Le segment fictif $[\mathsf{a}_6, \mathsf{b}_6]$ est ajoûté aux segments de la carte.

Le robot ne pouvant se trouver dans $\mathcal{P}_{\text{ext}}$, il est possible d'utiliser (4.21) pour réaliser un test sur les configurations. Considérons un point $\tilde{\mu}$ ayant pour coordonnées $(\tilde{x}, \tilde{y})$ dans $\mathcal{R}$. Lorsque la configuration du robot est $\mathsf{p} = (x_c, y_c, \theta)^{\mathrm{T}}$, en utilisant (4.1), on peut prouver aisément que les coordonnées de μ dans $\mathcal{W}$ dépendent de la configuration p. Lorsque μ est un point donné du robot, le test suivant permet d'éliminer des configurations pour lesquelles $\mu \notin \mathcal{P}_{\text{int}}$, c'est-à-dire pour lesquelles le robot ne serait donc pas entièrement dans $\mathcal{P}_{\text{int}}$.

Test $in\_room(\mu(\mathsf{p}))$:

$$\begin{cases} in\_room(\mu(\mathsf{p})) \text{ est vrai} & \text{si } \sum_{j=1}^{n_{\text{w}}} \arg\left(\overrightarrow{\mu(\mathsf{p})\,\mathsf{a}_j}, \overrightarrow{\mu(\mathsf{p})\,\mathsf{b}_j}\right) = 2\pi, \\ in\_room(\mu(\mathsf{p})) \text{ est faux} & \text{sinon.} \end{cases}$$

Lorsque μ est simplement la projection de p sur le plan (x, y), $in\_room(\mu(\mathsf{p}))$ est réécrit $in\_room(\mathsf{p})$.
♢

Le paragraphe 4.6 mettra en évidence la capacité de ce test à éliminer certaines configurations plus efficacement que lorsque le test d'association des données est utilisé seul. Notons que seules des conditions géométriques sont examinées, les mesures n'interviennent donc pas. Ce test, basé sur la parfaite connaissance de segments de la carte n'est utilisable en tant que tel que lorsque la partie de la carte ayant induit la partition de l'espace est fiable. Même lorsque ce n'est pas le cas, ce test demeure intéressant, car il forme la base du test *leg_in* présenté au paragraphe suivant et qui reste utilisable avec une carte erronée.

Remarque 4.3.3 *La figure 4-9 montre une pièce délimitée par* 4 *murs extérieurs, comportant un pilier, une caisse et un mur intérieur en L. Lorsque la taille et la position de ces* 3 *éléments est certaine, les segments les décrivant pourront être utilisés pour le test in_room. Par contre, en cas d'incertitude, seuls les murs extérieurs pourront servir au test.* ♢

4.3.3 Condition nécessaire de cohérence avec les données

Ce test va fournir une condition nécessaire pour la cohérence d'une configuration avec les mesures effectuées, plus facile à évaluer que le test complet dat_i.

Considérons une configuration $\mathsf{p} = (x_c, y_c, \theta)^{\mathrm{T}}$, le $i$ème capteur du robot $\tilde{\mathsf{s}}_i = (\tilde{x}_i, \tilde{y}_i)$, son cône d'émission $\mathcal{C}\left(\tilde{\mathsf{s}}_i, \tilde{\theta}_i, \tilde{\gamma}_i\right)$, et la mesure intervalle $[d_i]$ fournie par le capteur. Soit c_i le point situé à une distance $\underline{d_i}$ (borne inférieure de $[d_i]$) de $\tilde{\mathsf{s}}_i$ dans la direction d'émission $\tilde{\theta}_i$. Les coordonnées de c_i dans $\mathcal{W}$ satisfont

$$\mathsf{c}_i = \begin{pmatrix} x_c \\ y_c \end{pmatrix} + \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \tilde{x}_i + \cos\left(\tilde{\theta}_i\right) \underline{d_i} \\ \tilde{y}_i + \sin\left(\tilde{\theta}_i\right) \underline{d_i} \end{pmatrix}. \tag{4.23}$$

En considérant que $\mathcal{P}_{\text{int}}$ et $\mathcal{P}_{\text{ext}}$ ont été définis à partir de tous les segments de la carte, éventuellement complétés par des segments fictifs, on définit le test leg_in_i

Test $leg\_in_i(\mathsf{p})$: $\quad leg\_in_i(\mathsf{p}) = \overline{in\_room(\mathsf{s}_i(\mathsf{p}))} \vee in\_room(\mathsf{c}_i)$. ♢

La proposition suivante montre qu'il est possible d'utiliser ce test en renfort du test dat_i pour éliminer des configurations incohérentes avec les mesures. Si $leg\_in_i(\mathsf{p})$ est faux, c'est-à-dire si le capteur s_i est à l'intérieur de la pièce et le point c_i est à l'extérieur, alors nécessairement la $i$ème mesure ne pas peut être cohérente avec la carte :

Proposition 4 $leg\_in_i(\mathsf{p}) = 0 \Rightarrow dat_i(\mathsf{p}) = 0$. ♢

Preuve : $leg\_in_i(\mathsf{p}) = 0$ implique $in\_room(\mathsf{s}_i(\mathsf{p})) = 1$ et $in\_room(\mathsf{c}_i) = 0$; s_i appartient donc à $\mathcal{P}_{\text{int}}$ et c_i à $\mathcal{P}_{\text{ext}}$ (voir la figure 4-11). Il existe alors j tel que $[\mathsf{s}_i, \mathsf{c}_i] \cap [\mathsf{a}_j, \mathsf{b}_j] \neq \emptyset$ et $\mathsf{s}_i \in \Delta_{\mathsf{a}_j \mathsf{b}_j}$. Soit $\mathsf{m}_{ij} = [\mathsf{s}_i\mathsf{c}_i] \cap [\mathsf{a}_j, \mathsf{b}_j]$. Le $i$ème cône coupe $[\mathsf{a}_j, \mathsf{b}_j]$ au moins en m_{ij}. Ainsi l'espacement entre $[\mathsf{a}_j, \mathsf{b}_j]$ et s_i est inférieur ou égal à $\|\overrightarrow{\mathsf{s}_i\mathsf{m}_{ij}}\|$. Comme $\mathsf{m}_{ij} \in [\mathsf{s}_i\mathsf{c}_i]$ et $\mathsf{m}_{ij} \neq \mathsf{c}_i$, $\|\overrightarrow{\mathsf{s}_i\mathsf{m}_{ij}}\| < \|\overrightarrow{\mathsf{s}_i\mathsf{c}_i}\| = \underline{d_i}$, par conséquent l'espacement entre $[\mathsf{a}_j, \mathsf{b}_j]$ et s_i n'est pas compatible avec $[d_i]$, ce qui implique que $dat_i(\mathsf{p}) = 0$. ■

Le test $leg\_in_i(\mathsf{p})$ fournit ainsi une condition nécessaire pour que p soit cohérent avec la mesure fournie par le $i$ème capteur. Cette condition n'est cependant pas suffisante ; de ce fait, $leg\_in_i(\mathsf{p})$ peut valoir vrai même lorsque $dat_i(\mathsf{p})$ renvoie faux. Ce test ne sera donc utilisable que pour éliminer plus rapidement certaines configurations.

Remarque 4.3.4 *Lorsque des segments fictifs non réfléchissants sont employés pour compléter la carte, le test $leg\_in_i(\mathsf{p})$ doit être manipulé avec précaution. En effet, supposons que sur la figure 4-11, le segment $[\mathsf{a}_j, \mathsf{b}_j]$ soit fictif et représente le pas d'une porte ouverte. La mesure $[d_i]$ pourrait correspondre à une distance du capteur au battant de la porte ouverte. Or, le test leg_in_i considère cette configuration inacceptable. Dans de telles situations, il faudra tolérer un certain nombre de mesures aberrantes afin de ne pas éliminer abusivement certaines configurations, un tel traitement sera abordé au paragraphe 4.4.2.* ♢

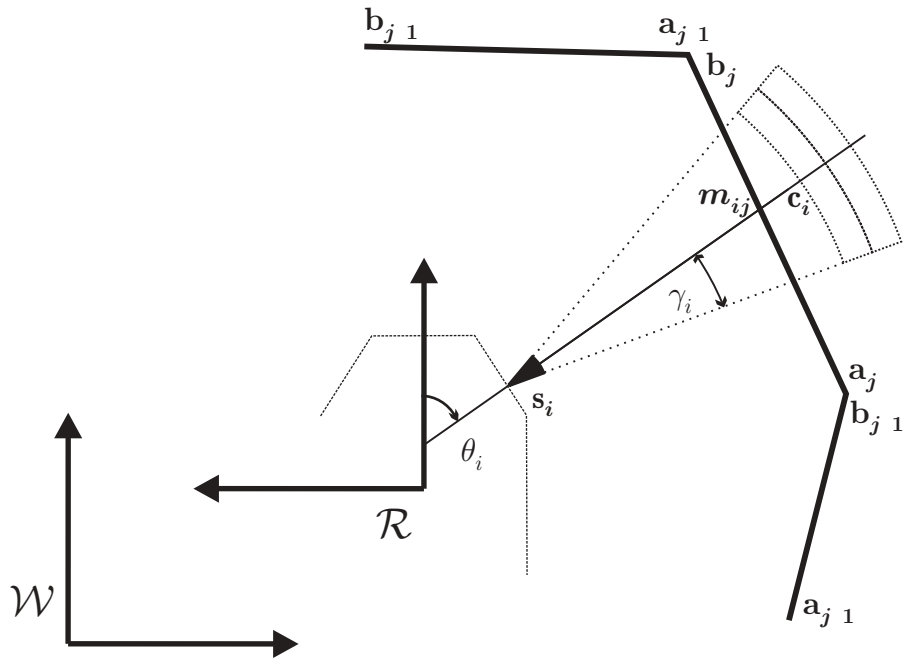


Figure 4-11: Le test leg_in_i vaut faux.

4.4 Tests d'inclusion

Les tests présentés dans les paragraphes précédents sont applicables à des configurations ponctuelles ; il s'agit maintenant d'en obtenir une version intervalle, afin de pouvoir les utiliser avec un algorithme d'inversion ensembliste pour tester des pavés de configurations. La notion d'intervalle booléen présentée au chapitre 2 sera utilisée pour définir une extension intervalle de ces tests. Nous verrons ensuite comment associer ces tests afin de renforcer leur efficacité.

4.4.1 Tests d'inclusion des tests de localisation

Un test d'inclusion naturel pour chaque test d'association de donnée dat_i est aisément construit en se servant de l'exemple 2.28. Considérons une configuration intervalle $[\mathsf{p}]$ donnée,

Test d'inclusion $dat_{i[]}([\mathsf{p}])$:

$$\begin{cases} dat_{i[]}([\mathsf{p}]) = 1 & \text{si } r_{i[]}([\mathsf{p}]) \subset [d_i]\,, \\ dat_{i[]}([\mathsf{p}]) = 0 & \text{si } r_{i[]}([\mathsf{p}]) \cap [d_i] = \emptyset, \\ dat_{i[]}([\mathsf{p}]) = [0,1] & \text{dans les autres cas.} \end{cases} \tag{4.24}$$

♢

Ce test repose sur l'évaluation de l'espacement, qui met en œuvre des branchements conditionnels. Pour de tels algorithmes, nous avons vu que la fonction χ présentée au paragraphe 2.7 permet de construire

une fonction d'inclusion. La version intervalle de l'évaluation de l'espacement est donnée en annexe B.2.

Le test d'inclusion naturel de *in_room* peut être très pessimiste, à cause de l'accumulation des incertitudes lorsque les sommes d'angles sont évaluées. Par conséquent, il est préférable d'utiliser la version intervalle spécifique de *in_room* suivante, où $[\mu]$ est le pavé encadrant l'ensemble $\mu([p])$ pour une configuration intervalle $[p]$ donnée,

Test d'inclusion $in\_room_{[]}([\mu])$:

$$\begin{cases} in\_room_{[]}([\mu]) = 1 & \begin{cases} \text{si } [a_j, b_j] \cap [\mu] = \emptyset, \text{ pour } j = 1, ..., n_w, \\ \text{et } in\_room(m([\mu])) = 1, \end{cases} \\ in\_room_{[]}([\mu]) = 0 & \begin{cases} \text{si } [a_j, b_j] \cap [\mu] = \emptyset, \text{ pour } j = 1, ..., n_w, \\ \text{et } in\_room(m([\mu])) = 0, \end{cases} \\ in\_room_{[]}([\mu]) = [0,1] & \text{dans les autres cas,} \end{cases}$$

avec $m([\mu])$ le centre de $[\mu]$. ♢

L'interprétation de ce test est très simple. Si $[\mu]$ ne coupe aucun segment de la carte, il est soit dans $\mathcal{P}_{\text{int}}$ soit dans $\mathcal{P}_{\text{ext}}$. Dans ces conditions, pour déterminer l'ensemble auquel $[\mu]$ appartient, il suffit de tester l'un des points de $[\mu]$ (par exemple $m([\mu])$). Comme au paragraphe 4.3, lorsque $[\mu]$ sera la projection de $[p]$ sur le plan (x, y), $in\_room_{[]}([\mu])$ sera écrit $in\_room_{[]}([p])$.

L'extension intervalle du test leg_in_i est obtenue en remplaçant simplement *in_room* par $in\_room_{[]}$.

Test d'inclusion $leg\_in_{i[]}([p])$: $\quad leg\_in_{i[]}([p]) = \overline{in\_room_{[]}(s_i([p]))} \vee in\_room_{[]}([c_i])$. ♢

4.4.2 Combinaison des tests de localisation

Les trois tests élémentaires définis au paragraphe 4.3 peuvent être combinés afin de former un test global $t(p)$ qui soit éventuellement plus fin que celui qu'on obtiendrait en utilisant uniquement dat_i. Dans le cas idéal où la carte fournie au robot est correcte et où toutes les mesures satisfont les hypothèses faites sur les erreurs de mesure, ce test global peut être écrit de la manière suivante

$$t_{\text{ideal}}(p) = in\_room(p) \wedge \left(\bigwedge_{i=1}^{n_s} dat_i(p) \right). \tag{4.25}$$

Une condition nécessaire pour que $dat_i(p)$ soit vrai est que $leg\_in_i(p)$ le soit. Comme cette condition n'est pas suffisante, leg_in_i ne peut être utilisé qu'en association avec dat_i afin d'éliminer plus facilement des configurations dans un contexte intervalle. Le test d'inclusion obtenu est alors

$$t_{\text{ideal}[]}([p]) = in\_room_{[]}([p]) \wedge \left(\bigwedge_{i=1}^{n_s} \left(\left(leg\_in_{i[]}([p]) \wedge [0,1] \right) \cap dat_{i[]}([p]) \right) \right). \tag{4.26}$$

D'après le théorème 2, ce test est plus fin que le test d'inclusion naturel de t_{ideal}.

Remarque 4.4.1 *Les tests élémentaires sont évalués de gauche à droite, ainsi, les tests les plus simples sont évalués en premier, afin de pouvoir éliminer le plus rapidement possible des configurations ne les satisfaisant pas. L'implantation réelle de ces tests est cependant légèrement différente de ce qu'indiquent les équations* (4.25) *et* (4.26)*. En effet, tous les tests* $leg\_in_{i[]}([\mathsf{p}])$*, dont l'évaluation est plus rapide que celle de* $dat_{i[]}([\mathsf{p}])$*, sont calculés en premier.* ♢

Supposons maintenant que la partie de la carte utilisée pour la définition de $\mathcal{P}_{\text{int}}$ reste vraie, mais que des mesures incohérentes sont présentes, qui ne satisfont pas les hypothèses faites sur les bornes des erreurs de mesure. Dans le contexte de la localisation de robot à partir de mesures télémétriques, la présence de telles mesures incohérentes est assez fréquente. Ces données peuvent correspondre, par exemple, à un phénomène de réflexion multiple, à l'absence d'onde réfléchie, à la présence d'êtres humains ou d'objets non cartographiés dans l'environnement du robot, à un mauvais fonctionnement de capteurs, etc. En présence de telles données aberrantes, l'ensemble $\mathcal{S}$, défini par t_{ideal} peut se révéler vide. Grâce à la fonction *et-q-relaxé* $\bigoplus_q$ introduite au paragraphe 2.7.1, $t_{\text{ideal}[]}$ peut être modifié en

$$t_{\text{outliers}[]}([\mathsf{p}], q) = in\_room_{[]}([\mathsf{p}]) \wedge \left(\bigoplus_{i=1}^{n_s}{}_{q[]} \left(\left(leg\_in_{i[]}([\mathsf{p}]) \wedge [0,1] \right) \cap dat_{i[]}([\mathsf{p}]) \right) \right), \tag{4.27}$$

afin de tolérer jusqu'à q données aberrantes. Une approche possible de l'estimation robuste aux données aberrantes est de commencer avec $q = 0$, ce qui correspond à l'utilisation d'un test équivalent à t_{ideal}, puis d'incrémenter q d'une unité tant que l'ensemble des configurations cohérentes avec $n_s - q$ données reste vide. Plus de détails sur cette technique et sur le critère d'arrêt peuvent être trouvés dans [Jaulin et al.96] ou [Jaulin et al.98]. Ces articles présentent une version garantie de l'estimateur Omne (Outlier Minimal Number Estimator - estimateur correspondant au nombre minimum de données aberrantes, voir [Lahanier et al.87], [Walter et al.88], et [Pronzato et al.96]).

Lorsque $\mathcal{P}_{\text{int}}$ et $\mathcal{P}_{\text{ext}}$ ne sont pas fiables (c'est-à-dire lorsque la carte est susceptible de comporter des erreurs), le test $in\_room_{[]}$ ne doit plus être considéré dans $t_{\text{ideal}[]}$ ou dans $t_{\text{outliers}[]}$, suivant la confiance accordée aux autres données disponibles. Une autre possibilité, qui ne sera pas considérée par la suite est d'accorder la même confiance au test $in\_room_{[]}$ qu'au test $dat_{i[]}$ et de définir un nouveau test

$$t_{\text{robust}[]}([\mathsf{p}], q) = \bigoplus_{i=0}^{n_s}{}_{q[]} \left(t_{i[]}([\mathsf{p}]) \right), \tag{4.28}$$

avec

$$\begin{aligned} &t_{0[]}([\mathsf{p}]) = in\_room_{[]}([\mathsf{p}]), \\ &t_{i[]}([\mathsf{p}]) = \left(leg\_in_{i[]}([\mathsf{p}]) \wedge [0,1] \right) \cap dat_{i[]}([\mathsf{p}]), \quad i = 1, \ldots, n_s. \end{aligned} \tag{4.29}$$

L'objectif du paragraphe suivant est de montrer comment à l'aide de Sivia et des tests d'inclusion définis précédemment il est possible de caracteriser l'ensemble des configurations compatibles avec les mesures disponibles et les informations *a priori* d'une manière systématique.

4.5 Inversion ensembliste

Nous avons vu un certain nombre de tests permettant de déterminer si une configuration est ou non cohérentes avec les mesures et les informations *a priori*. La caractérisation de l'ensemble des configurations cohérentes à l'intérieur d'un pavé de recherche initial $[\mathbf{p}_0]$ peut parfaitement être réalisée de manière systématique et garantie grâce à l'algorithme MaskSivia présenté au chapitre 3. L'ensemble $\mathcal{S} = \{\mathbf{p} \in [\mathbf{p}_0] \mid t(\mathbf{p}) = 1\}$ est caractérisé de manière approchée mais garantie par un sous-pavage intérieur et extérieur.

Dans le paragraphe suivant, nous allons examiner le comportement de l'algorithme de localisation dans différentes situations. Les différents tests présentés au paragraphe 4.4.2 seront appliqués, avec et sans vecteur de masque. Le comportement de l'algorithme à l'égard de données aberrantes sera également examiné.

4.6 Applications

La localisation utilisant les techniques d'analyse par intervalles qui viennent d'être présentées sera illustrée par trois exemples caractéristiques. Bien qu'utilisant des données simulées, ces exemples demeurent très réalistes ; les caractéristiques du robot (taille, position des capteurs et performances) sont celles du modèle représenté sur la figure 4-1. Ce robot dispose d'une ceinture périphérique de $n_s = 24$ capteurs à ultrasons. On détermine expérimentalement (voir [Leveque98]) leur demi-angle d'émission au sommet $\tilde{\gamma} = 0.2\ rad$ et leur précision relative de mesure $\alpha = 2\%$ à l'intérieur des bornes de fonctionnement du capteur $[0.1\text{ m}, 10.7\text{ m}]$. La portée limitée des capteurs n'a pas été prise en compte ici car elle n'interviendrait que de manière très marginale dans les exemples proposés.

Dans chacun des trois cas traités, le domaine de recherche initial dans l'espace des configurations est $[-12\text{ m}, 12\text{ m}] \times [-12\text{ m}, 12\text{ m}] \times [0, 2\pi]$. MaskSivia utilise une bissection des pavés suivant la dimension de plus grande longueur pondérée, le vecteur de poids est dans les trois cas $\left(1\text{ m}^{-1}, 1\text{ m}^{-1}, 1\text{ rad}^{-1}\right)$ et le paramètre de précision ϵ est pris égal à 0.04. Seul le sous-pavage extérieur est considéré. Tous les calculs sont réalisés sur un Pentium 233MMX, en utilisant l'implantation en C++ de MaskSivia que nous avons développée.

Remarque 4.6.1 *Le vecteur de poids doit être choisi de manière à ce que l'incertitude sur θ ait les mêmes conséquences sur la position du robot que l'incertitude sur x_c et y_c. Si une incertitude $\delta x_c \approx \delta y_c$ est tolérée sur la position du milieu de l'essieu des roues motrices, $\delta\theta$ doit être telle que l'incertitude sur la position d'un point μ quelconque du robot δx_μ ou δy_μ soit de l'ordre de δx_c. Or $\delta x_\mu \approx \delta y_\mu \approx R_\mu \delta\theta$*

où R_μ est la distance entre c *et* μ*. Par conséquent, pour obtenir l'incertitude maximale sur x_μ et y_μ, il suffit de prendre un majorant de la distance entre* c *et les autres points du robot, appelé le* rayon R *du robot. Ici, on prendra $R = 1$ m et le vecteur de poids est $\left(\frac{1}{R}, \frac{1}{R}, 1\right)$.* ♢

Remarque 4.6.2 *Le choix du paramètre de précision ϵ est en général assez délicat. Il a une influence sur le temps de calcul et sur la précision avec laquelle l'ensemble solution sera décrit. Avec ϵ diminuant, le temps de calcul et la précision de description augmentent. Un compromis possible consiste à choisir un ϵ qui fournit une precision de description de la configuration qui soit de l'ordre de grandeur de l'erreur sur les mesures.* ♢

4.6.1 Premier exemple : comparaison des tests

Ce premier exemple n'a pour objectif que de démontrer la contribution des différentes procédure d'accélération présentées dans ce chapitre dans des conditions idéales (carte exacte, pas de données aberrantes). Le robot est placé dans une pièce qui est ici supposée décrite exactement par la carte représentée sur la figure 4-12. La figure 4-13 décrit le *diagramme d'émission* des 24 capteurs : pour chaque capteur i, on représente en pointillés sa direction d'émission et deux arcs correspondant à l'intersection de son cône d'émission et de deux cercles de rayons $\underline{d_i}$ et $\overline{d_i}$. Au moins un obstacle doit se trouver au moins partiellement entre ces deux arcs.

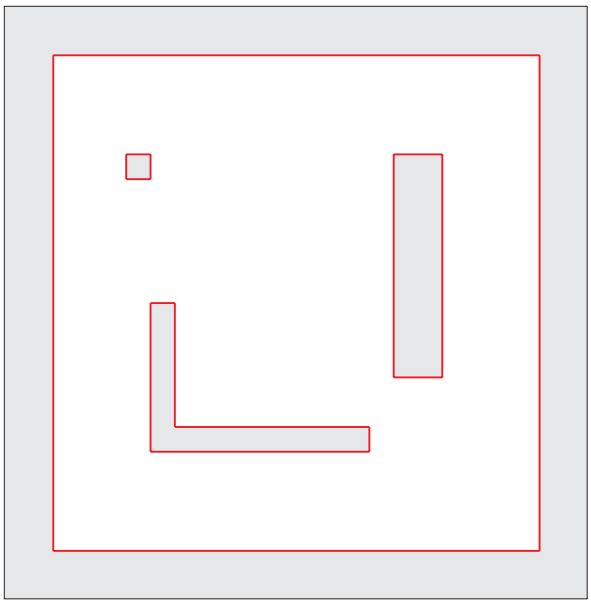


Figure 4-12: Carte fournie au robot pour les trois exemples. La projection du pavé de recherche initial sur le plan (x, y) est matérialisée par le carré extérieur.

Ce diagramme est obtenu grâce à l'algorithme décrit dans l'annexe B.1 pour la configuration du robot $(x_c, y_c, \theta) = \left(-2, 3, \frac{9\pi}{32}\right)$. Cette "vraie" configuration n'est évidemment pas transmise à l'algorithme

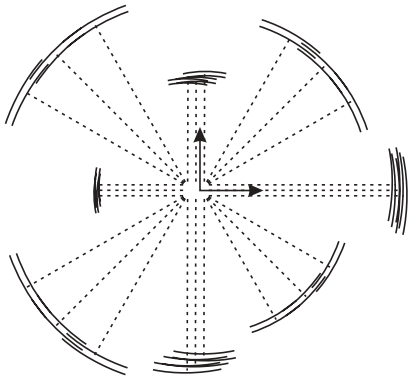


Figure 4-13: Diagramme d'émission (premier exemple).

de localisation du robot. Le tableau 4.1 présente les temps de calcul obtenus pour la localisation avec diverses combinaisons des tests présentés. Dans tous les cas, l'ensemble solution varie très peu ; la figure 4-14 présente l'ensemble des configurations obtenu par l'algorithme complet. L'union des pavés présentés contient de manière garantie l'ensemble des configurations compatibles avec la carte et les mesures effectuées. La véritable configuration est incluse dans le pavé noir.

test	masque	temps (s)
$t_{\mathrm{dat}[]}$	non	97
$t_{\mathrm{dat}[]}$	oui	29
$t_{\mathrm{room}[]}$	non	91
$t_{\mathrm{room}[]}$	oui	27
$t_{\mathrm{leg}[]}$	non	48
$t_{\mathrm{leg}[]}$	oui	11
$t_{\mathrm{ideal}[]}$	non	49
$t_{\mathrm{ideal}[]}$	oui	11

Tableau 4.1: Temps de calcul pour le premier exemple.

Le test $t_{\mathrm{dat}[]}$ ne met en œuvre que le test $dat_{i[]}, i = 1, \ldots, n_{\mathrm{s}}$. Le test $t_{\mathrm{room}[]}$ combine $in\_room_{[]}$ et $t_{\mathrm{dat}[]}$. Le test $t_{\mathrm{leg}[]}$ utilise $leg\_in_{i[]}, i = 1, \ldots, n_{\mathrm{s}}$ pour renforcer $t_{\mathrm{dat}[]}$. Enfin, $t_{\mathrm{ideal}[]}$ rassemble l'ensemble de ces tests de la manière décrite au paragraphe 4.4.2.

Sur cet exemple, la version de MaskSivia (cf. chapitre 3) utilisant les tests masqués $t_{\mathrm{leg}[]}$ ou $t_{\mathrm{ideal}[]}$ est environ 10 fois plus rapide que la version standard non masquée de Sivia utilisant seulement $t_{\mathrm{dat}[]}$. Le masque semble responsable de la majeure partie de l'amélioration, suivi du test $leg\_in_{i[]}$ et de $in\_room_{[]}$. Lorsque le masque et le test $t_{\mathrm{leg}[]}$ sont utilisés, $in\_room_{[]}$ n'apporte pas d'amélioration sensible, mais nous gardons à l'esprit que les tests $leg\_in_{i[]}$ utilisent $in\_room_{[]}$.

Afin de mettre en évidence le rôle du masque dans chacun des tests élémentaires, nous avons mis en place une version de Sivia employant $t_{\mathrm{ideal}[]}$ où les masques sont initialisés sans être utilisés pour réduire le nombre de calculs à effectuer. Le tableau 4.2 indique le nombre de fois qu'un test est appliqué et le nombre de fois que l'usage du masque permettrait d'éviter un calcul pour déterminer la valeur du test.

test	nb appels	nb utilisation du masque	utilisation du masque (%)
in_room	1175	591	50 %
leg_in	1175	635	54 %
leg_in_i	39887	29741	74 %
dat_i	31527	10457	33 %
r_{ij}	546516	475575	87 %

Tableau 4.2: Influence du masque.

En moyenne, le masque permet de réduire de moitié le nombre de calculs à effectuer. La version masquée fait intervenir un calcul de l'espacement un peu différent de la version présentée. Plutôt que de calculer tous les $[r_{ij}]$ et d'en prendre le plus petit pour définir $[r_i]$ et le comparer ensuite à $[d_i]$, chaque $[r_{ij}]$ est comparé à $[d_i]$, ce qui permet de stocker les résultats de cette comparaison dans un masque. Cette technique fournit d'excellent résultats, car elle permet de diviser par neuf le nombre d'évaluations nécessaires de l'éloignement.

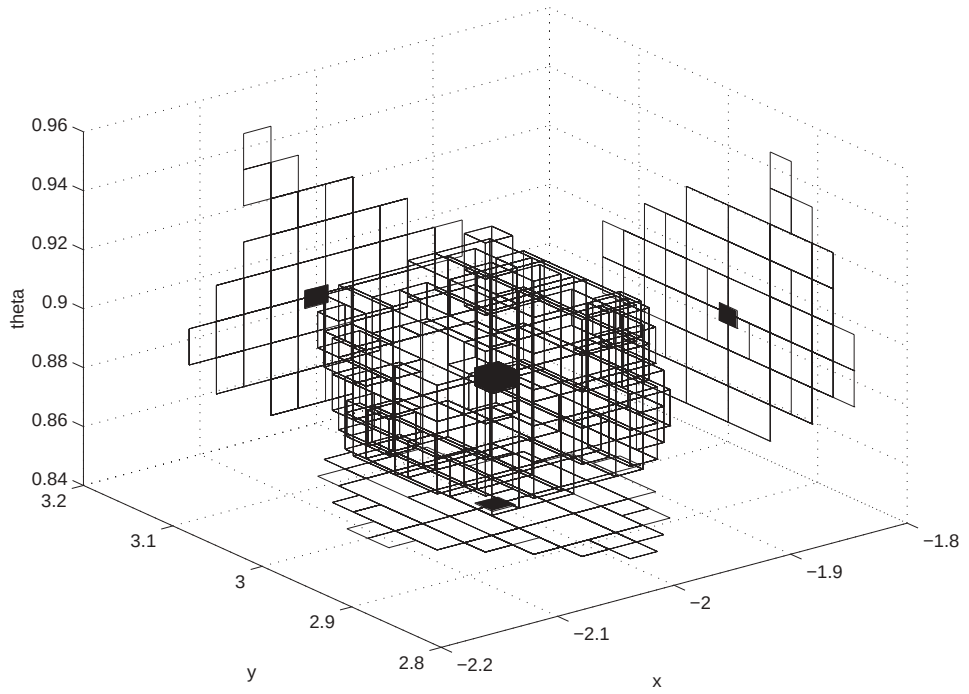


Figure 4-14: Approximation extérieure de l'ensemble de toutes les configurations et projections (premier exemple). La configuration réelle appartient au pavé noir.

Les deux exemples suivant présentent deux situations où la localisation est plus difficile mais également plus réaliste.

4.6.2 Second exemple : configurations ambiguës

La pièce et la carte restent identiques à celles décrites dans le premier exemple, cependant, la véritable configuration est maintenant $(x_c, y_c, \theta) = (-1, -7.5, \pi)$, et le diagramme d'émission est représenté sur la figure 4-15.

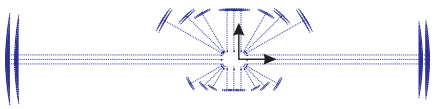


Figure 4-15: Diagramme d'émission (Exemple test 2).

En 19 secondes, Sivia utilisant $t_{\text{ideal}[]}$ masqué fournit l'ensemble de pavés représenté sur la figure 4-16. Cet ensemble est composé de deux parties disjointes, dont l'une contient la véritable configuration du robot.

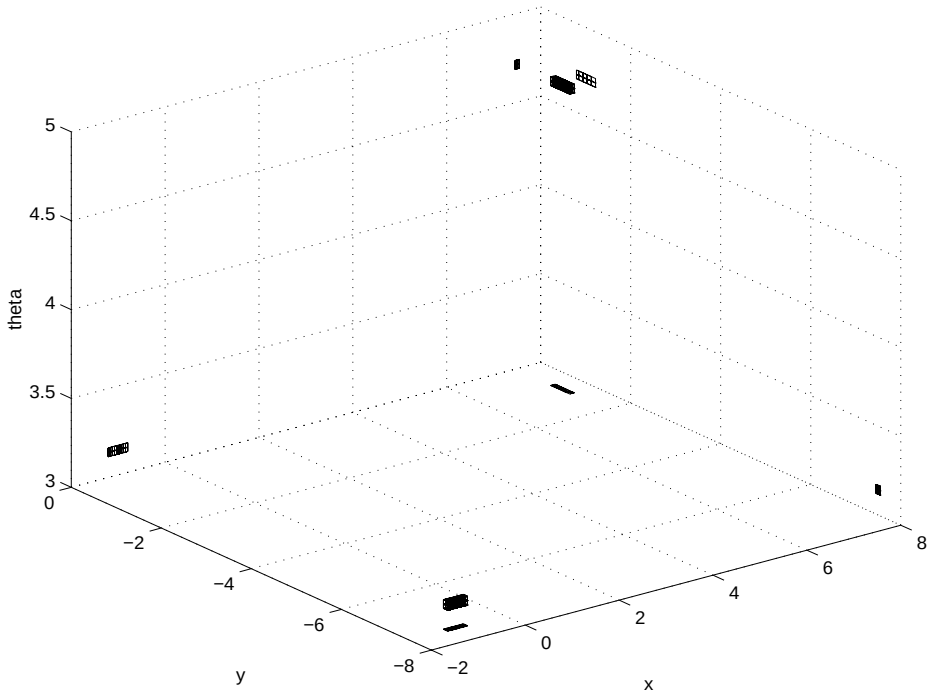


Figure 4-16: Approximation extérieure de l'ensemble des configurations possibles (second exemple) et projections.

La figure 4-17 illustre les raisons de cette ambiguïté. A cause d'une symétrie locale de la pièce, deux types de configurations très différents sont possibles ; chacun correspond à une association spécifique des segments de la carte aux distances mesurées par les capteurs. Notons que cette association des mesures aux segments est directement fournie par l'algorithme (il suffit d'examiner le masque) et n'est pas une

étape nécessaire dans la procédure de localisation, comme l'imposent d'autres méthodes (voir par exemple [Drumheller87]). Comme cette étape d'association est un point très délicat dans la localisation autonome, la méthode de localisation utilisant l'analyse par intervalles présente ici un avantage non négligeable.

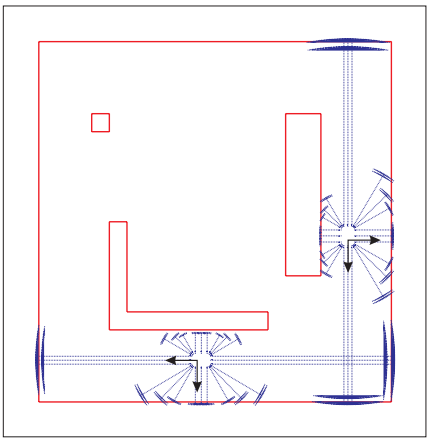


Figure 4-17: Deux configurations possibles (second exemple).

4.6.3 Troisième exemple : présence de données aberrantes

Lorsque des données aberrantes sont présentes, la localisation devient bien plus complexe. Cette difficulté est introduite dans ce dernier exemple test. La carte dont dispose le robot est la même que dans les deux tests précédents, mais elle est maintenant partiellement incorrecte. Le véritable environnement du robot est représenté sur la figure 4-18. Par comparaison avec la figure 4-12, on constate que le pilier a été déplacé, et qu'un second pilier a été ajouté. De plus, 2 distances mesurées sur les 24 ont été doublées afin de simuler un phénomène de réflexion multiple. La véritable configuration est la même que dans le premier exemple test. L'une quelconque des modifications introduites dans cet exemple (par exemple la carte incorrecte ou les données aberrantes) est suffisante pour que l'ensemble fourni par l'algorithme non modifié soit vide.

Comme la carte est incorrecte, le test $in\_room_{[]}$ ne pourra pas être utilisé. Le nombre de données aberrantes q est incrémenté jusqu'à ce que l'ensemble fourni par le test $t_{\text{outliers}[]}$ sans $in\_room_{[]}$ devienne non-vide, ce qui est le cas à partir de $q = 6$. L'ensemble des configurations possibles est de ce fait légèrement plus grand que celui représenté sur la figure 4-14, mais en reste très proche et ne sera donc pas représenté. Cet ensemble contient toujours la véritable configuration du robot. La figure 4-19 illustre une configuration type appartenant à cet ensemble. Toutes les mesures qui n'ont pas pu être associés à un segment ont été numérotées. Les cônes d'émission 1 et 6 ne sont pas cohérents avec la carte, à cause de la présence d'obstacles plus proches des capteurs correspondants. Les cônes d'émission 2 à 5 résultent de deux piliers mal représentés sur la carte. Le tableau 4.3 indique les temps de calcul et différentes propriétés de l'ensemble solution $\widehat{\mathcal{S}}$ en fonction de q.

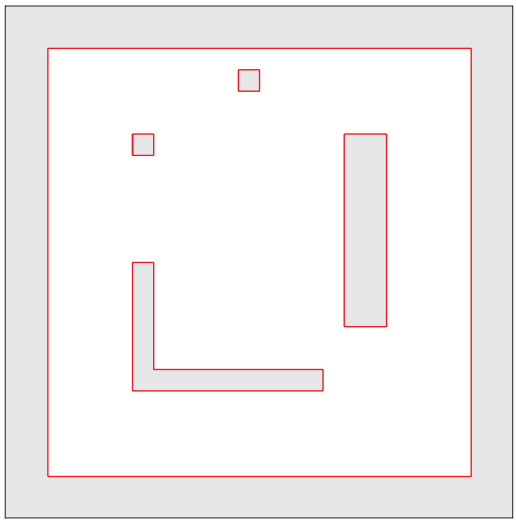


Figure 4-18: Pièce réelle du troisième exemple.

Notons que l'ensemble de pavés obtenu pour une valeur donnée de q ne contient la véritable configuration du robot de manière garantie que s'il n'y a effectivement pas plus de q données aberrantes. Il est possible de se protéger contre un nombre plus important de données aberrantes en augmentant q. Dans ce cas, les ensembles obtenus pour $q > 6$ sont assez proches de celui obtenu avec $q = 6$, le nombre de données aberrantes effectif.

q	volume de $\mathcal{S}$	pavé extérieur (m, m, rad)	temps (s)	temps cumulé (s)
0	0	$\emptyset$	7	7
1	0	$\emptyset$	14	21
2	0	$\emptyset$	20	41
3	0	$\emptyset$	30	71
4	0	$\emptyset$	42	113
5	0	$\emptyset$	53	166
6	2.68×10^{-3}	$[-2.14, -1.87][2.85, 3.15][0.83, 0.95]$	83	249
7	3.09×10^{-3}	$[-2.14, -1.87][2.85, 3.15][0.83, 0.95]$	117	366
8	4.25×10^{-3}	$[-2.16, -1.82][2.83, 3.17][0.83, 0.95]$	153	519
9	5.88×10^{-3}	$[-2.18, -1.82][2.83, 3.19][0.83, 0.96]$	257	776
10	8.05×10^{-3}	$[-2.21, -1.80][2.81, 3.19][0.82, 0.97]$	350	1126

Tableau 4.3: Caracteristiques de $\widehat{\mathcal{S}}$ et temps de calcul en fonction de q (troisième exemple).

Le temps de calcul augmente avec q. Ceci est dû au fait qu'il devient de plus en plus difficile d'éliminer une configuration à mesure que l'on augmente le nombre de données aberrantes toléré. La figure 4-20 montre que l'évolution du volume de $\widehat{\mathcal{S}}$ ne présente pas de véritable saut lorsque q est plus grand que la valeur à partir de laquelle $\widehat{\mathcal{S}}$ devient non-vide. De ce fait, elle ne fournit pas nécessairement de critère de choix de q.

De plus, choisir un q supérieur à la valeur $q_{\min}$ telle que $\mathcal{S} \neq \emptyset$ permet d'augmenter la robustesse par rapport à d'éventuelles données aberrantes non détectées qui sont toujours possibles.

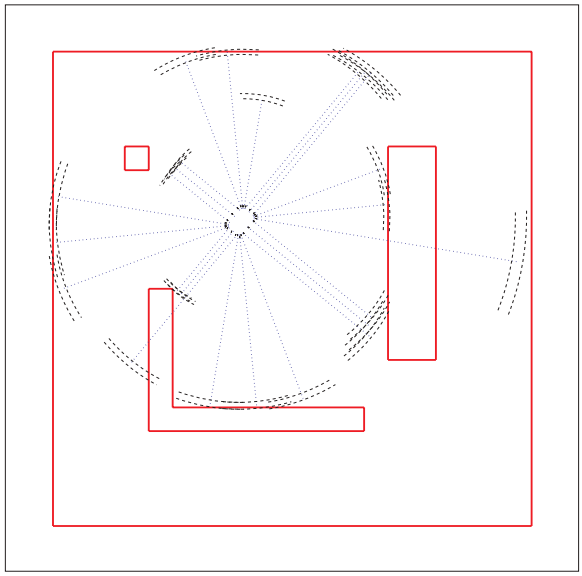


Figure 4-19: Une configuration possible pour le troisième exemple.

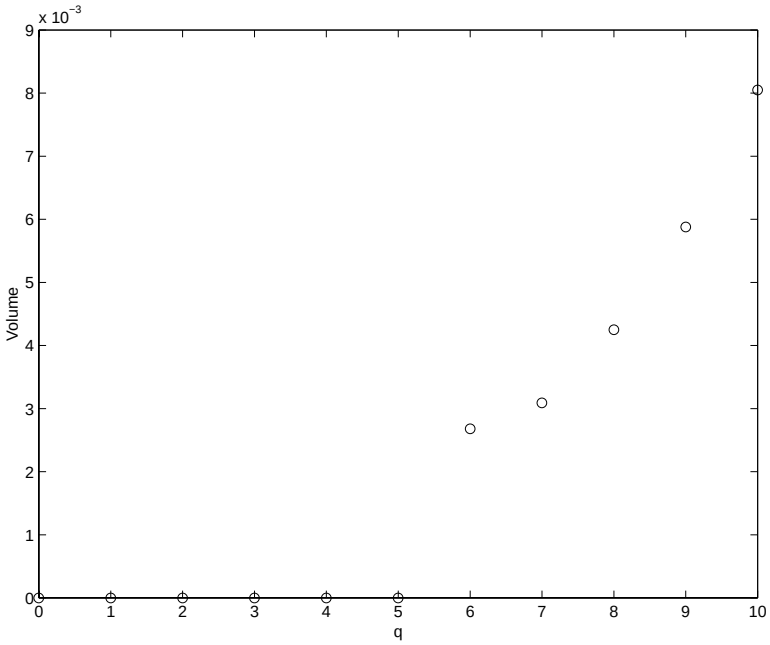


Figure 4-20: Volume du sous-pavage en fonction de q

Une fois de plus, rappelons que contrairement aux méthodes nécessitant une phase d'association des données, il n'est pas nécessaire de réaliser un pré-traitement pour décider quelles sont les q mesures parmi les n_s qui doivent être considérées comme incohérentes ; ceci à nouveau constitue un simplification significative.

4.7 Conclusions

Le problème de la localisation autonome d'un robot est particulièrement adapté à une résolution utilisant l'analyse par intervalles à cause du faible nombre de paramètres à estimer. Dans ce contexte, la méthode proposée dans ce chapitre présente quelques avantages significatifs sur ses concurrentes. Il n'est pas nécessaire d'examiner l'ensemble des combinaisons possibles entre données fournies par les capteurs et éléments de la carte, pas plus qu'il n'est nécessaire, le cas échéant de choisir q données aberrantes parmi n_s points de mesure. Par conséquent, l'explosion combinatoire intervenant dans d'autres algorithmes est évitée. Les résultats obtenus sont globaux et aucune configuration compatible avec les mesures et les informations *a priori* ne peut être oubliée. L'estimateur est extrêmement robuste et fournit des résultats corrects, même avec un nombre très important de données aberrantes. En outre, ses résultats demeurent garantis tant que le nombre réel de données aberrantes reste inférieur ou égal à la tolérance q fixée lors de la localisation.

Cette méthode est très souple, car de nouveaux tests tenant compte d'une meilleure connaissance physique du problème peuvent aisément être ajouté afin d'affiner ceux déjà implantés. Il est possible, par exemple, de tenir compte du fait que la portée des capteurs à ultrasons est limitée, ou que l'angle d'incidence des ondes émises doit rester suffisamment petit pour que l'onde réfléchie ou diffractée par un obstacle puisse être reçue par le capteur.

De nombreux autres tests élémentaires pourraient également être pris en compte. Par exemple, il serait facile de modifier le test leg_in_i pour prendre en compte non seulement le point $\mathtt{c}_i$ mais également tout autre point appartenant au cône d'émission et se trouvant à une distance inférieure à $\underline{d_i}$ du capteur. Il reste cependant à voir si l'augmentation de la complexité des calculs permettrait encore de réduire le temps de calcul global par une diminution du nombre de configurations à tester.

Dans ce chapitre, la localisation est statique ; une extension naturelle de ce travail est la localisation dynamique d'un robot mobile, en effectuant un suivi de l'ensemble de ses configurations possibles. Nous verrons aux chapitres 5 et 6 que l'analyse par intervalles peut également apporter des solutions à ce problème plus complexe.

Cette méthodologie s'applique évidemment à de nombreux autres domaines où la faisabilité s'exprime sous forme d'inégalités non-linéaires. Le cas où certaines de ces inégalités apparaissent comme incohérentes peut parfaitement être traité de la même manière que les données aberrantes ont été traitées ici.

Chapitre 5

Estimation d'état de systèmes non-linéaires discrets

5.1 Introduction

L'estimation d'état a pour but de suivre, généralement en temps réel, le comportement de variables internes d'un dispositif à partir de mesures de ses entrées et de ses sorties. Il convient, dans un premier temps de définir ces variables qui seront représentatives, à un instant donné, de la situation physique du système et formeront le vecteur d'état. Une seconde étape consiste à obtenir une *équation d'évolution* de l'état en fonction du temps et des entrées appliquées au système. A partir d'un état initial connu, cette équation permettra de prédire les réponses futures à des entrées connues. Le vecteur d'état pourra être de plus ou moins grande dimension, suivant la complexité du modèle considéré pour décrire le système à analyser.

Pour tenir compte du fait que le modèle ne correspond pas exactement au système réel, il est possible d'ajoûter un *bruit d'état* à l'équation d'évolution de l'état. Souvent, l'état n'est pas directement mesurable, par contre, un certain nombre de grandeurs physiques seront mesurées et elles permettront d'estimer l'état par calcul. L'*équation d'observation* décrit le processus qui à partir d'un état donné fournit les grandeurs mesurées. Pour tenir compte des incertitudes de mesure et pour éventuellement tenir compte du fait que l'équation d'observation ne décrit pas exactement la relation entre état et mesure, un *bruit de mesure* peut être ajouté.

Ces deux équations permettent, grâce par exemple au filtrage de Kalman dans le cas d'un modèle linéaire, de prédire l'état à l'aide de l'équation d'évolution, puis de corriger cette estimée à l'aide de l'équation d'observation.

Ce chapitre va aborder le thème de l'estimation d'état de systèmes non-linéaires discrets, sous l'hypothèse d'erreurs bornées sur l'état et sur les mesures. La présence de paramètres inconnus, voire variables dans le temps, sera également considérée par introduction d'un vecteur d'état étendu aux paramètres.

Le problème est, pour un modèle déterminé, de caractériser à un instant donné et de manière garantie l'*ensemble* des vecteurs d'état compatibles avec les sorties mesurées jusqu'alors et les hypothèses faites sur les bruits de mesure et d'état. Généralement, dans le contexte de l'estimation à erreur bornée, le modèle est supposé linéaire (ou il est linéarisé), et des ellipsoïdes ou des parallélotopes sont évalués à chaque pas d'échantillonnage ; ces ensembles contiennent de manière garantie l'état à l'instant d'évaluation, voir par exemple [Schweppe68], [Schweppe73], [Maksarov et al.96], [Durieu et al.96a] et [Durieu et al.96b]. Les algorithmes mis en œuvre s'inspirent du schéma du filtrage de Kalman, alternant phases de prédiction et de correction. Dans le cas de modèles non linéaires, les problèmes sont beaucoup plus difficiles si on veut conserver le caractère garanti des résultats. En effet, l'ensemble contenant les états cohérents avec les informations disponibles à un instant donné peut ne pas être convexe, ni même connexe. Nous souhaitons pourtant obtenir une caractérisation garantie de cet ensemble (elle ne sera qu'approchée, mais avec une précision arbitraire). Un schéma récursif sera présenté, afin de faciliter une mise en œuvre des algorithmes en temps réel. Avant de passer à la présentation de la technique développée, mentionnons que d'autres auteurs ont employé l'analyse par intervalles pour construire un pendant intervalle au filtre de Kalman pour des modèles linéaires incertains [Chen et al.97]. Cependant l'estimateur qu'ils ont obtenu ne garantit nullement que l'état reste bien dans le pavé qu'il fournit. L'algorithme que nous avons développé est quant à lui applicable à des systèmes non-linéaires, éventuellement incertains, et garantit ses résultats.

Une procédure d'estimation d'état idéalisée sera présentée au paragraphe 5.2, ainsi qu'un exemple illustratif. Un outil de description d'ensemble, le *sous-pavage*, sera introduit au paragraphe 5.3. Cet outil sera ensuite utilisé pour bâtir une version approchée, mais réalisable en pratique, de l'algorithme d'estimation d'état non-linéaire récursif. Les étapes de correction et de prédiction seront présentées aux paragraphes 5.4 et 5.5. Le paragraphe 5.6 fait la synthèse de l'ensemble de ces notions pour l'algorithme d'estimation d'état et traite en détail l'exemple illustratif introduit au paragraphe 5.2. Enfin, la mise en œuvre informatique des sous-pavages sera présentée au paragraphe 5.7 (voir également [Kieffer et al.98a] et [Kieffer et al.98b]).

5.2 Estimation d'état

Soit le système non linéaire discret et éventuellement variant dans le temps décrit par les équations d'évolution et d'observation suivantes

$$\begin{cases} \mathsf{x}_{k+1} = \mathsf{f}_k\left(\mathsf{x}_k, \mathsf{u}_k, \mathsf{v}_k\right), \\ \mathsf{y}_k = \mathsf{h}_k\left(\mathsf{x}_k\right) + \mathsf{w}_k, \end{cases} \quad k = 0, 1, \ldots \tag{5.1}$$

où $\mathsf{u}_k \in \mathbf{R}^m, \mathsf{x}_k \in \mathbf{R}^n$ et $\mathsf{y}_k \in \mathbf{R}^p$ sont respectivement les vecteurs d'entrée, d'état et de sortie. L'état initial x_0 est supposé appartenir à un ensemble compact $\mathcal{X}_0 \subset \mathbf{R}^n$. Les suites inconnues $\{\mathsf{v}_k\}$ et $\{\mathsf{w}_k\}$ décrivent les bruits d'état et de mesure (ou d'observation) ; la seule hypothèse faite sur ces bruits est leur appartenance à des suites de pavés connues $\{[\mathsf{v}]_k\}$ et $\{[\mathsf{w}]_k\}$. Les fonctions (ou les algorithmes finis)

f_k et h_k évaluant x_{k+1} et y_k à chaque pas k sont connues.

Nous nous intéresserons à l'évaluation récursive du plus petit ensemble $\mathcal{X}_l$ contenant de manière garantie toutes les valeurs de l'état x_l au pas l compatibles avec les informations disponibles à l'instant l, c'est-à-dire avec

$$\mathcal{I}_l = \left\{ \mathcal{X}_0, \{\mathsf{u}_k, \mathsf{y}_k, [\mathsf{v}]_k, [\mathsf{w}]_k\}_{k=0}^{l} \right\}. \tag{5.2}$$

Remarque 5.2.1 *Pour simplifier la présentation, le bruit de mesure a été supposé additif, ce qui n'est pas restrictif car aucune hypothèse d'indépendance entre le bruit d'état et l'état n'a été faite. D'autres structures peuvent également être considérées, à condition qu'elles permettent l'écriture d'une relation entre vecteurs d'état et de sortie du type* $\mathsf{g}(\mathsf{y}_k, \mathsf{w}_k) = \mathsf{h}_k(\mathsf{x}_k)$. ♢

Remarque 5.2.2 *Le problème plus général de l'estimation conjointe de l'état et des paramètres peut être traité dans ce contexte. Il suffit de remplacer* x_k *par un vecteur d'*état étendu $\mathsf{x}_k^e = \left(\mathsf{x}_k^T, \mathsf{p}_k^T\right)^T$ *incorporant le vecteur de paramètres inconnus* $\mathsf{p}_k \in \mathbf{R}^q$*, supposé appartenir à un ensemble compact* $\mathcal{P}_0 \subset \mathbf{R}^q$ *connu* a priori. *Une équation d'évolution pour le vecteur des paramètres est également requise ; cette équation peut par exemple être du type* $\mathsf{p}_{k+1} = \mathsf{p}_k + \mathsf{v}_k^p$*, où* $\{\mathsf{v}_k^p\}$ *est une suite de vecteurs inclus dans les pavés de la suite* $\{[\mathsf{v}^p]_k\}$*, suite également connue* a priori. *L'état étendu vérifie alors*

$$\left\{ \begin{array}{l} \mathsf{x}_{k+1}^e = \left(\begin{array}{c} \mathsf{x}_{k+1} \\ \mathsf{p}_{k+1} \end{array} \right) = \left(\begin{array}{c} \mathsf{f}_k(\mathsf{x}_k^e, \mathsf{u}_k, \mathsf{v}_k, \mathsf{v}_k^p) \\ \mathsf{p}_k + \mathsf{v}_k^p \end{array} \right) = \mathsf{f}_k^e(\mathsf{x}_k^e, \mathsf{u}_k, \mathsf{v}_k^e), \\ \mathsf{y}_k = \mathsf{h}_k^e(\mathsf{x}_k^e) + \mathsf{w}_k, \end{array} \right. \quad k = 0, 1, ... \tag{5.3}$$

avec $\mathsf{v}_k^e = \left(\mathsf{v}_k^T, (\mathsf{v}_k^p)^T\right)^T \in [\mathsf{v}^e]_k = \left([\mathsf{v}]_k^T, [\mathsf{v}^p]_k^T\right)^T$ *et* $\mathsf{x}_0^e \in \mathcal{X}_0^e = \mathcal{X}_0 \times \mathcal{P}_0 \subset \mathbf{R}^n \times \mathbf{R}^q$*. Toute information complémentaire éventuellement disponible sur la dépendance du vecteur de paramètres* p_{k+1} *en* p_k *et* x_k *peut être introduite dans l'équation* (5.3). ♢

5.2.1 Algorithme idéalisé

Soit $\mathcal{X}_l$ l'ensemble des valeurs de x_l compatibles avec $\mathcal{I}_l$, et $\mathcal{X}_{l+}$ l'ensemble des valeurs de l'état accessibles à partir d'un état quelconque x_l dans $\mathcal{X}_l$ lorsque l'entrée est u_l et le bruit d'état $\mathsf{v}_l \in [\mathsf{v}]_l$. $\mathcal{X}_{l+}$ est donc donné par

$$\begin{array}{rcl} \mathcal{X}_{l+} & = & \{\mathsf{f}_l(\mathsf{x}, \mathsf{u}_l, \mathsf{v}_l) \mid \mathsf{x} \in \mathcal{X}_l, \mathsf{v}_l \in [\mathsf{v}]_l\} \\ & = & \mathsf{f}_l(\mathcal{X}_l, \mathsf{u}_l, [\mathsf{v}]_l). \end{array} \tag{5.4}$$

Soit $\mathcal{Y}_{l+1}$ l'ensemble des valeurs que peut prendre la sortie lorsque sa valeur mesurée est y_{l+1}. Il a pour expression

$$\mathcal{Y}_{l+1} = \left\{ \mathsf{y}_{l+1} - \mathsf{w}_{l+1} \mid \mathsf{w}_{l+1} \in [\mathsf{w}]_{l+1} \right\} = \mathsf{y}_{l+1} - [\mathsf{w}_{l+1}] \,. \tag{5.5}$$

Soit enfin $\mathcal{X}^o_{l+1}$ l'ensemble de toutes les valeurs de l'état qui peuvent avoir conduit à une observation $\mathsf{y} \in \mathcal{Y}_{l+1}$

$$\mathcal{X}^o_{l+1} = \{ \mathsf{x} \in \mathbf{R}^n \mid \mathsf{h}_{l+1}(\mathsf{x}) \in \mathcal{Y}_{l+1} \} = \mathsf{h}^{-1}_{l+1}(\mathcal{Y}_{l+1}) \,. \tag{5.6}$$

L'ensemble $\mathcal{X}_{l+1}$ de toutes les valeurs de x_{l+1} compatibles avec $\mathcal{I}_{l+1}$ est alors donné par l'intersection des ensembles définis par (5.4) et (5.6).

$$\mathcal{X}_{l+1} = \mathcal{X}_{l+} \cap \mathcal{X}^o_{l+1}. \tag{5.7}$$

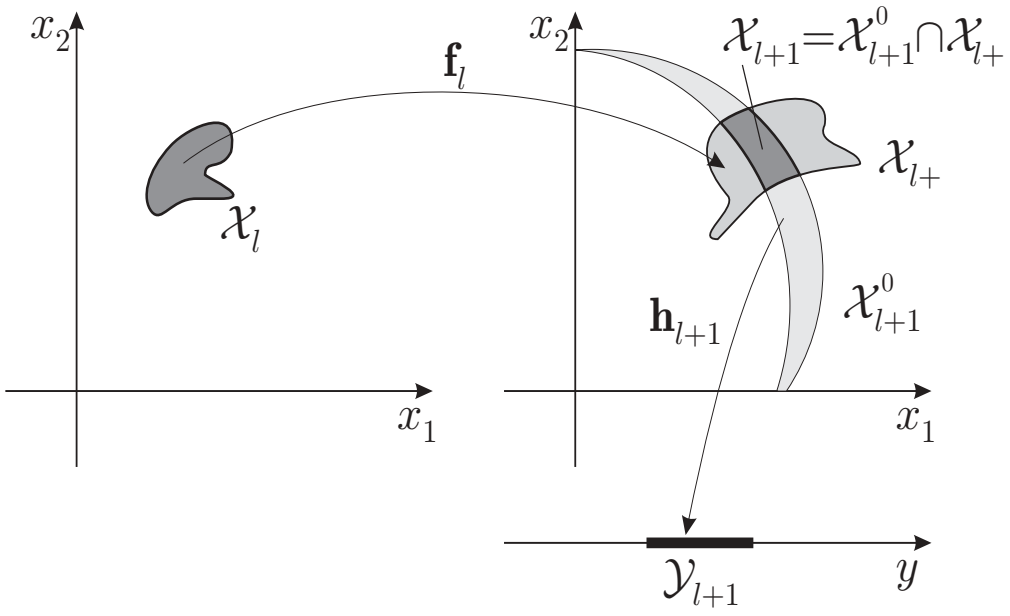


Figure 5-1: Principe de l'algorithme d'estimation d'état idéalisé. $\mathcal{X}_l$ est l'état à l'instant l. $\mathcal{X}_{l+}$ est l'état prédit à l'instant $l+1$. Une mesure $\mathcal{Y}_{l+1}$ est faite à $l+1$, $\mathcal{X}^0_{l+1}$ est l'ensemble des états ayant pu conduire à une mesure appartenant à $\mathcal{Y}_{l+1}$. Par conséquent, l'état à l'instant $l+1$ est $\mathcal{X}^0_{l+1} \cap \mathcal{X}_{l+}$.

La figure 5-1 illustre les différentes étapes de l'algorithme d'estimation d'état idéalisé. Comme $\mathcal{X}_0$ contient toutes les valeurs du vecteur d'état initial x_0, il est possible, au moins théoriquement, d'évaluer de façon récursive l'ensemble $\mathcal{X}_l$, contenant toutes les valeurs de l'état au pas l, compatibles avec les informations disponibles. Les étapes de cet algorithme idéalisé sont résumées dans l'algorithme suivant

Algorithme 2 *Pour $l = 0$ à L effectuer*

1. *Prédiction* : $\mathcal{X}_{l+} = \mathsf{f}_l\left(\mathcal{X}_l, \mathsf{u}_l, [\mathsf{v}]_l\right)$.

2. *Correction* : $\mathcal{X}_{l+1} = \mathsf{h}_{l+1}^{-1}\left(\mathcal{Y}_{l+1}\right) \cap \mathcal{X}_{l+}$.

Théorème 5.2.1 *$\mathcal{X}_l$, évalué par l'algorithme 2, est le plus petit ensemble contenant les valeurs x_l compatibles avec $\mathcal{I}_l$.* ♢

Preuve : la démonstration est par récurrence et par contradiction. Pour $l = 0$, le théorème 5.2.1 est vrai par définition de $\mathcal{X}_0$. Supposons qu'il soit vrai pour $\mathcal{X}_k$ et que $\mathcal{X}_{k+1}$ soit évalué par l'algorithme 2. Supposons qu'il existe $\mathcal{X}'_{k+1} \subset \mathcal{X}_{k+1}$ contenant toutes les valeurs possibles de x_{k+1} et qu'il existe un état $\tilde{\mathsf{x}} \in \mathcal{X}_{k+1}$ tel que $\tilde{\mathsf{x}} \notin \mathcal{X}'_{k+1}$. Comme $\tilde{\mathsf{x}} \in \mathcal{X}_{k+1}$, $\tilde{\mathsf{x}} \in \mathsf{f}_k\left(\mathcal{X}_k, \mathsf{u}_k, [\mathsf{v}]_k\right)$, il existe alors $\mathsf{x}_k \in \mathcal{X}_k$ et $\mathsf{v}_k \in [\mathsf{v}]_k$ tel que $\tilde{\mathsf{x}} = \mathsf{f}_k\left(\mathsf{x}_k, \mathsf{u}_k, \mathsf{v}_k\right)$. De plus $\mathsf{h}_{k+1}\left(\tilde{\mathsf{x}}\right) = \mathsf{h}_{k+1}\left(\mathsf{f}_k\left(\mathsf{x}_k, \mathsf{u}_k, \mathsf{v}_k\right)\right) \in \mathsf{y}_{k+1} - [\mathsf{w}]_{k+1}$, de ce fait il existe $\mathsf{w}_{k+1} \in [\mathsf{w}]_{k+1}$ tel que $\mathsf{h}_{k+1}\left(\tilde{\mathsf{x}}\right) + \mathsf{w}_{k+1} = \mathsf{y}_{k+1}$. Ainsi $\tilde{\mathsf{x}}$ est une image possible de $\mathsf{x}_k \in \mathcal{X}_k$ et $\tilde{\mathsf{x}} \in \mathcal{X}'_{k+1}$, ce qui contredit l'hypothèse initiale. ■

Sauf dans des cas très particuliers, il n'est pas possible de donner une caractérisation exacte des ensembles $\mathcal{X}_{l+}$, et $\mathcal{X}_{l+1}$. Notre objectif sera donc de donner une approximation *extérieure* de $\mathcal{X}_{l+1}$, qui soit aussi précise que possible. Avant de décrire les blocs composant l'algorithme qui réalisera cette tâche, présentons un exemple illustratif qui permettra en particulier d'expliciter l'équation d'évolution et l'équation d'observation de (5.1).

5.2.2 Exemple de la balle au rebond

Considérons une balle rebondissant sur le sol. Estimer l'état d'un tel système n'est pas aussi simple qu'il y paraît, car le rebond provoque une discontinuité de l'équation d'évolution décrivant le mouvement de la balle. Ce problème illustre une difficulté typique rencontrée par exemple lorsqu'on cherche à décrire le mouvement d'une patte de robot marcheur ou dans de nombreux systèmes hybrides [Hyb93]. La solution présentée dans la suite de ce chapitre peut être aisément étendue à des systèmes plus complexes, comme nous le verrons au chapitre 6.

Le mouvement est supposé unidimentionnel, et la balle incompressible de rayon r. De ce fait, l'altitude par rapport au sol x_1 et par la vitesse $x_2 = \dot{x}_1$ constituent l'état de la balle. Nous étudions l'évolution de l'état $\mathsf{x} = (x_1, x_2)^{\mathrm{T}}$ de la balle lorsqu'elle est lâchée avec pour état initial x_0, qu'on sait appartenir à un ensemble $\mathcal{X}_0$ connu. Des mesures bruitées de la hauteur de la balle sont effectuées à intervalle de temps réguliers T. L'équation d'observation a donc la forme suivante

$$y_k = h_k\left(\mathsf{x}_k\right) + w_k = (1\ 0)\,\mathsf{x}_k + w_k,\ k = 0, 1, ...$$

avec w_k appartenant à un intervalle connu.

En négligeant les frottements et la compression de la balle au rebond, le mouvement peut être décrit par

$$\begin{cases} \dot{x}_1 = x_2, \\ \dot{x}_2 = -g, \end{cases}$$

pendant la chute, et

$$\begin{cases} x_1 \rightarrow x_1, \\ x_2 \rightarrow -x_2, \end{cases}$$

à l'instant du rebond. A l'aide d'une discrétisation exacte, il est possible de construire un algorithme fini évaluant x_{k+1} pour un état x_k donné, voir l'annexe E.1.

5.3 Sous-pavages

Les étapes de prédiction et de correction de l'algorithme 2 peuvent être mises en œuvre de manière approchée mais garantie à l'aide de l'analyse par intervalles et de la notion de *sous-pavage*, qui permet une description efficace des ensembles compacts de $\mathbf{R}^n$. Cette section a pour but de présenter les sous-pavages, la manière dont ils sont organisés et quelques propriétés de ces sous-ensembles de $\mathbf{R}^n$. La réalisation informatique de cette structure ainsi que le détail de certains algorithmes de traitement seront abordés au paragraphe 5.7.

Un sous-pavage $\widehat{\mathcal{X}}$ de dimension n est une union de pavés disjoints de $\mathbf{R}^n$. L'ensemble de tous les sous-pavages de $\mathbf{R}^n$ est noté $\mathcal{K}(\mathbf{R}^n)$. Nous allons montrer qu'il est possible d'approximer, avec une précision arbitraire, tout sous-ensemble compact de $\mathbf{R}^n$ à l'aide d'un sous-pavage. Pour évaluer la distance entre un compact et un sous-pavage de $\mathbf{R}^n$, la distance de Hausdorff $d(.,.)$ introduite au chapitre 2 est utilisée.

Proposition 5 *Soit $(\mathcal{C}(\mathbf{R}^n), \subset, d)$ l'ensemble des compacts de $\mathbf{R}^n$, muni de la relation d'ordre partiel $\subset$ et de la distance de Hausdorff $d(.,.)$. Pour tout élément $\mathcal{A} \in \mathcal{C}(\mathbf{R}^n)$ et pour tout $\epsilon > 0$, il existe $\widehat{\mathcal{X}_\epsilon} \in \mathcal{K}(\mathbf{R}^n)$ tel que $\mathcal{A} \subset \widehat{\mathcal{X}_\epsilon}$ et $d\left(\mathcal{A}, \widehat{\mathcal{X}_\epsilon}\right) < \epsilon$.* ♢

Une preuve relativement formelle d'une proposition analogue peut être trouvée dans [Berger79a]. La démonstration suivante est plus algorithmique, mais ne fait intervenir que les propriétés de la distance de Hausdorff.

Preuve : *soit $\mathcal{A} \in \mathcal{C}(\mathbf{R}^n)$ et $\epsilon > 0$. Comme $\mathcal{A}$ est un compact, c'est un ensemble borné et il existe un pavé $[\mathsf{x}]_{\mathcal{A}} \subset \mathbf{R}^n$ tel que $\mathcal{A} \subset [\mathsf{x}]_{\mathcal{A}}$. Soit $\widehat{\mathcal{X}_{\mathcal{A}}^0}$ un sous-pavage construit par bissections successives du pavé $[\mathsf{x}]_{\mathcal{A}}$ en sous-pavés $[\mathsf{x}]_i$, $i = 1, ..., N$, dont la longueur maximale est inférieure à ϵ. Supposons que $\widehat{\mathcal{X}_\epsilon}$ soit construit à partir de l'algorithme suivant*

Pour $i = 1$ à N

si $d_0\left([\mathsf{x}]_i, \mathcal{A}\right) > \epsilon$, d'après le lemme2.6.3, $[\mathsf{x}]_i \cap A = \emptyset$,

créer un nouveau sous-pavage approximant

$\widehat{\mathcal{X}_{\mathcal{A}}^i}$ valant $\mathcal{X}_{\mathcal{A}}^{[-1}$ privé de $[\mathsf{x}]_i$,

sinon $\widehat{\mathcal{X}_{\mathcal{A}}^i} = \mathcal{X}_{\mathcal{A}}^{[-1}$.

$\widehat{\mathcal{X}_\epsilon} = \widehat{\mathcal{X}_{\mathcal{A}}^N}$.

Pour chaque $i = 1, ..., N$, $\mathcal{A} \subset \widehat{\mathcal{X}_{\mathcal{A}}^i}$, *ainsi d'après le lemme 2.6.1,* $d_0\left(\mathcal{A}, \widehat{\mathcal{X}_\epsilon}\right) = 0$. *De plus,* $d_0\left(\widehat{\mathcal{X}_\epsilon}, \mathcal{A}\right) = \max_{\mathsf{x} \in \widehat{\mathcal{X}_i}} d_0\left(\mathsf{x}, \mathcal{A}\right) = \max_{[\mathsf{x}]_i \in \widehat{\mathcal{X}_i}} \left(\max_{\mathsf{x} \in [\mathsf{x}]_i} d_0\left(\mathsf{x}, \mathcal{A}\right)\right) < \epsilon$ *par construction, et ainsi* $d\left(\mathcal{A}, \widehat{\mathcal{X}_\epsilon}\right) < \epsilon$. ■

Comme les sous-pavages sont des ensembles de pavés, nous allons voir qu'il est possible d'étendre les opérations sur les intervalles aux sous-pavages. La manière de décrire ces derniers aura un impact déterminant sur la complexité de ces opérations.

Pour illustrer les différentes manières de décrire un sous-pavage, considérons le cas bidimentionnel de la figure 5-2. L'ensemble $\mathcal{X}$ (en noir), inclus dans le pavé $[\mathsf{x}_0] = [0, 8]^2$, est représenté par le sous-pavage $\widehat{\mathcal{X}}$ (en gris).

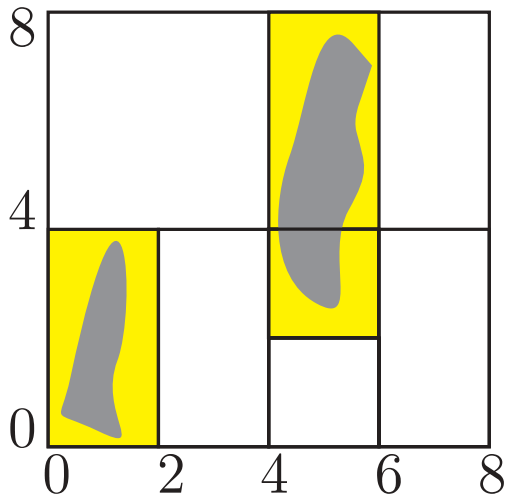


Figure 5-2: $\mathcal{X}$ (en noir) représenté par le sous-pavage $\widehat{\mathcal{X}}$ (en gris).

5.3.1 Description par des listes

Les pavés contenus dans le sous-pavage $\widehat{\mathcal{X}}$ représenté sur la figure 5-2, peuvent être simplement énumérés dans une liste

$$\mathcal{L} = \{[0, 2] \times [0, 4], [4, 6] \times [2, 4], [4, 6] \times [4, 8]\}.$$

Cependant, comme tous ces pavés sont obtenus à partir d'une succession de bissections du pavé initial $[\mathsf{x}_0]$, il est également possible de stocker le pavé initial et la manière dont il a été coupé pour obtenir chacun des pavés de la liste. Il est nécessaire pour cela d'introduire une notation pour spécifier la manière dont un pavé est bissecté, et désigner la partie sélectionnée pour réaliser la bissection suivante. Si $[\mathsf{x}]$ est un pavé de dimension n

$$L_j\left[\mathsf{x}\right] = \left(\left[x\right]_1, ..., \left[x\right]_{j-1}, \left[\underline{x_j}, c\left(\left[x\right]_j\right)\right], \left[x\right]_{j+1}, ..., \left[x\right]_n\right)$$

et

$$R_j\left[\mathsf{x}\right] = \left(\left[x\right]_1, ..., \left[x\right]_{j-1}, \left[c\left(\left[x\right]_j\right), \overline{x_j}\right], \left[x\right]_{j+1}, ..., \left[x\right]_n\right)$$

seront respectivement les *sous-pavés gauche* et *droit* obtenus par bissection de la $j^{\text{ème}}$ composante de $[\mathsf{x}]$. En utilisant récursivement cette notation, il est possible de réécrire $\mathcal{L}$ de la manière suivante

$$\mathcal{L} = \left\{L_1L_2L_1\left[\mathsf{x}_0\right],\ R_2L_1L_2R_1\left[\mathsf{x}_0\right],\ L_1R_2R_1\left[\mathsf{x}_0\right]\right\}.$$

Evidemment, cette description n'est pas unique. Ainsi, par exemple, $R_1L_2L_1\left[\mathsf{x}_0\right] = L_2R_1L_1\left[\mathsf{x}_0\right]$. Cette ambiguïté peut être levée en considérant une règle de bissection canonique. Dans la suite de ce document, la règle de bissection canonique retenue est de couper la composante de *longueur maximale* (voir le paragraphe 2.6) de chaque pavé $[\mathsf{x}] \subset \mathbf{R}^n$, mais d'autres règles pourraient aisément être utilisées.

Un sous-pavage sera dit *régulier* si chacun des pavés le composant a été obtenu en appliquant la règle de bissection canonique. Pour des sous-pavages réguliers et à l'aide de la règle canonique, il est très facile de retrouver la dimension suivant laquelle s'est faite la bissection. Il n'est donc plus nécessaire d'indicer L et R. Par conséquent, $\mathcal{L}$ peut être réécrit comme suit

$$\left\{LLL\left[\mathsf{x}_0\right], RLLR\left[\mathsf{x}_0\right], LRR\left[\mathsf{x}_0\right]\right\}.$$

La règle de bissection choisie n'aura aucune influence sur la garantie des résultats fournis par les algorithmes ; par contre, l'efficacité, les temps de calcul risquent d'être modifiés, voir par exemple le paragraphe 3.2.2.

5.3.2 Description par des arbres binaires

Nous avons vu que lorsque les algorithmes du chapitre 3 ne pouvaient statuer sur l'état d'un pavé, ce pavé *père* était bissecté en deux pavés *enfants*. Si, après traitement, il s'avère que les pavés enfants appartiennent tous deux à l'ensemble solution, il peut être opportun, afin de réduire le nombre de pavés de cet ensemble, de les réunir à nouveau et de les remplacer par leur père. Pour cela, il faut stocker l'information sur la relation existant entre deux enfants d'un même père, ce que ne permet pas facilement

la structure de rangement en liste.

Un *arbre binaire ordonné* [Beidler96] est une structure idéale pour stocker ce type de dépendance. Un arbre binaire $\widehat{\mathcal{X}}$ contient un ensemble fini de *nœuds*. Cet ensemble peut être vide, ne contenir qu'un seul nœud, la *racine* de l'arbre binaire, ou encore deux arbres binaires disjoints, les *sous-arbres gauche* et *droit*. Ainsi, l'arbre binaire de la figure 5-3 représente le sous-pavage $\widehat{\mathcal{X}}$ de la figure 5-2.

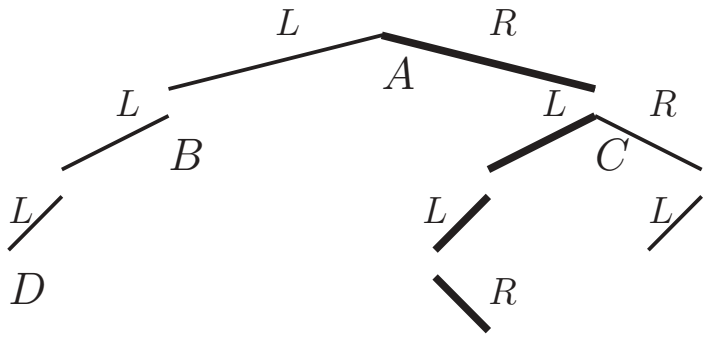


Figure 5-3: Arbre binaire représentant le sous-pavage $\widehat{\mathcal{X}}$ de la figure 5-2.

A est la racine de l'arbre. B et C sont respectivement le *fils gauche* et le *fils droit* de A. Ce sont des nœuds *frères* car ils ont le même *père* A. En outre, A a un sous-arbre gauche et un sous-arbre droit ; B a un sous-arbre droit *vide*. A et B sont des nœuds, car ils disposent au moins d'un sous-arbre non vide. Enfin, comme aucun sous-arbre ne part de D, c'est un nœud dégénéré ou une *feuille*.

L'arbre binaire représentant un sous-pavage peut être construit par exemple à l'aide de la liste $\mathcal{L}$ de ses pavés. La croissance des branches est déterminée par la manière dont le pavé initial $[\mathsf{x}_0]$, correspondant à la racine de l'arbre, est bissecté. La présence d'un nœud indique que le pavé correspondant a été bissecté suivant la règle de bissection canonique. Une feuille indique que le pavé correspondant appartient au sous-pavage. Par exemple, la branche marquée en gras sur la figure 5-3 correspond au pavé $RLLR\,[\mathsf{x}_0] = [4,6]\times[2,4]$. La *profondeur* d'un pavé indique le nombre de bissection qui ont été nécessaires pour l'obtenir à partir du pavé racine. Ainsi, la profondeur du pavé $[4,6]\times[2,4]$ est 4. La profondeur d'un sous-pavage est la plus grande profondeur de ses pavés.

Un arbre (sous-pavage) est dit *minimal* s'il ne dispose pas de feuilles (pavés) frères. En effet, un sous-pavage dont l'arbre représentatif comporte de telles feuilles issues d'un même père demeure identique lorsque son arbre est simplifié, en retirant les fils, le père devenant une feuille.

Les notions d'arbre binaires et de sous-pavage réguliers étant équivalentes, le vocabulaire des arbres sera utilisé également pour les sous-pavages. Dans la suite de ce document, la représentation des pavés par des arbres binaires sera adoptée à cause de la récursivité naturelle de cette structure de données, cependant, l'équivalence entre un arbre binaire et la liste des pavés qu'il représente ne doit pas être oubliée.

Il est maintenant possible de décrire des compacts de $\mathbf{R}^n$ de manière approchée par des sous-pavages.

Pour manipuler de tels objets, deux algorithmes principaux ont été développés : un algorithme d'inversion ensembliste et un algorithme de calcul d'image directe d'un ensemble par une fonction. Le paragraphe 5.7 traite de la description informatique des arbres binaires adaptés aux sous-pavages et du codage des opérations sur ces ensembles. Le fait de savoir qu'il est possible de réaliser une inversion ensembliste et un calcul d'image permet maintenant de construire un algorithme d'estimation d'état récursive approchée. Nous allons dans les sections suivantes nous attacher à la construction des étapes de correction et de prédiction requises par l'algorithme 2, en commençant par l'étape de correction car c'est la plus simple.

5.4 Correction garantie

L'étape de correction consiste à caractériser l'ensemble

$$\mathcal{X}_{k+1} = \{\mathsf{x} \in \mathcal{X}_{k+} \mid \mathsf{h}_{k+1}(\mathsf{x}) \in \mathcal{Y}_{k+1}\} \tag{5.8}$$

par une approximation extérieure. Cette tâche peut être placée dans le cadre des problèmes d'*inversion ensembliste*, comme ceux traités au paragraphe 3.4. En effet, (5.8) peut se récrire

$$\mathcal{X}_{k+1} = \mathsf{h}_{k+1}^{-1}(\mathcal{Y}_{k+1}) \cap \mathcal{X}_{k+}. \tag{5.9}$$

Ce type de problème est résolu de manière approchée grâce à Sivia. Cet algorithme fournit deux listes de pavés (qui ne sont autre chose que des sous-pavages) l'une contenue et l'autre contenant de manière garantie l'ensemble à évaluer. Une version récursive de Sivia, mieux adaptée à un traitement sur les sous-pavages et permettant en particulier de réaliser l'inversion d'un sous-pavage, va maintenant être présentée.

Supposons que $\mathcal{X}$ et $\mathcal{Y}$ soient respectivement approchés par les sous-pavages extérieurs $\widehat{\mathcal{X}}$ et $\widehat{\mathcal{Y}}$. L'objectif d'un algorithme d'inversion ensembliste est d'approcher l'ensemble

$$\mathcal{S} = \mathsf{f}_{\mathcal{X}}^{-1}(\mathcal{Y}) = \{\mathsf{x} \in \mathcal{X} \mid \mathsf{f}(\mathsf{x}) \in \mathcal{Y}\}, \tag{5.10}$$

par un sous-pavage extérieur

$$\widehat{\mathcal{S}} = \mathsf{f}_{\widehat{\mathcal{X}}}^{-1}\left(\widehat{\mathcal{Y}}\right) = \left\{\mathsf{x} \in \widehat{\mathcal{X}} \mid \mathsf{f}(\mathsf{x}) \in \widehat{\mathcal{Y}}\right\}. \tag{5.11}$$

Lemme 5.4.1 *Soient $\mathcal{S}$ et $\widehat{\mathcal{S}}$ définis respectivement par* (5.10) *et* (5.11). *On a alors $\mathcal{S} \subset \widehat{\mathcal{S}}$* ♢

Preuve : Soit $\mathsf{x}_0 \in \mathcal{S}$. On a $\mathsf{x}_0 \in \mathcal{X}$ et comme $\mathcal{X} \subset \widehat{\mathcal{X}}$, $\mathsf{x}_0 \in \widehat{\mathcal{X}}$; de plus $\mathsf{f}(\mathsf{x}_0) \in \mathcal{Y}$, comme $\mathcal{Y} \subset \widehat{\mathcal{Y}}$, $\mathsf{f}(\mathsf{x}_0) \in \widehat{\mathcal{Y}}$. Par conséquent $\mathsf{x}_0 \in \widehat{\mathcal{S}}$. ■

L'algorithme récursif SiviaSp (Sivia pour les Sous-pavages) est utilisé pour réaliser cette tâche. Comme on peut le prévoir, une exploration des nœuds de $\widehat{\mathcal{X}}$ sera réalisée à l'aide d'une fonction d'inclusion

$\mathsf{f}_{[]}$ de f.

Soit $[\mathsf{x}]$ le pavé correspondant à un nœud donné N de $\widehat{\mathcal{X}}$. Si $\mathsf{f}_{[]}([\mathsf{x}]) \subset \widehat{\mathcal{Y}}$, le sous-pavage issu de N est entièrement inclus dans l'ensemble $\mathcal{S}$ à caractériser, et de ce fait, il doit être stocké dans le sous-pavage $\widehat{\mathcal{S}}$. Si $\mathsf{f}_{[]}([\mathsf{x}]) \cap \widehat{\mathcal{Y}} = \emptyset$, alors $[\mathsf{x}] \cap \mathcal{S} = \emptyset$, ainsi, N et le sous-pavage dont il est la racine ne sont plus à considérer. Si le résultat des deux tests précédents est négatif et si $w_{\infty}([\mathsf{x}]) > \epsilon$ (on peut aussi utiliser $w_{\mathrm{rel}\infty}([\mathsf{x}]) > \epsilon$ ou $w_{\mathrm{p}\infty}([\mathsf{x}], \mathsf{p}) > \epsilon$), les sous-pavages issus de N doivent être testés à leur tour, sinon le pavé correspondant à N est considéré comme étant suffisamment petit et est stocké dans $\widehat{\mathcal{S}}$. Le réel positif ϵ est choisi par l'utilisateur et détermine la précision souhaitée dans la description de l'ensemble à caractériser.

Exemple 5.4.1 *Considérons le sous-pavage $\widehat{\mathcal{X}}$ représenté en gris sur la colonne centrale de la figure 5-4. Nous allons chercher le sous-ensemble $\mathcal{P}$ de $\widehat{\mathcal{X}}$ défini de la manière suivante*

$$\mathcal{P} = \left\{ (x, y) \in \widehat{\mathcal{X}} \,|x > 0 \right\}.$$

Ce problème peut être formulé comme un problème d'inversion ensembliste

$$\mathcal{P} = f^{-1}(]0, +\infty[) \cap \widehat{\mathcal{X}},$$

avec

$$f(x, y) = x.$$

Le comportement de l'algorithme SiviaSp *est représenté sur la figure 5-4.*
Les nœuds déjà examinés sont représentés par un cercle noir, celui en cours d'analyse est représenté par un point noir sur la colonne de gauche. Le pavé correspondant est encadré en gras sur le sous-pavage, l'évolution de la solution potentielle apparaît sur la droite. Dès l'étape (2)*, nous constatons que de larges parties de l'arbre sont très rapidement éliminées. A l'étape* (6)*, la précision limite n'ayant pas été atteinte, on fait croître les branches de l'arbre, ce qui revient à opérer une bissection sur des pavés du sous-pavage correspondant aux feuilles. A l'étape* (8)*, les deux sous-pavés frères sont solution, il sont donc réunis, le pavé père devenant solution.* ♢

Le code complet de SiviaSp est donné dans la section 5.7. Ainsi, $\mathsf{SiviaSp}(\widehat{\mathcal{X}}, \mathsf{f}_{[]}, \widehat{\mathcal{Y}}, \epsilon)$ fournit un sous-pavage contenant $\mathsf{f}_{\widehat{\mathcal{X}}}^{-1}\left(\widehat{\mathcal{Y}}\right)$.

Exemple 5.4.2 *Considérons les sous-pavages $\widehat{\mathcal{A}} = \left\{ [-3, 3]^2 \right\}$ et $\widehat{\mathcal{B}_1} = \{[2, 4]\}$, ainsi que la fonction $f_1(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2$. L'ensemble $\mathcal{C}_1 \subset \widehat{\mathcal{A}}$ tel que $f_1(\mathcal{C}_1) = \widehat{\mathcal{B}_1}$ est contenu de manière garantie dans le sous-pavage $\widehat{\mathcal{C}_1} = \mathsf{SiviaSp}(\widehat{\mathcal{A}}, f_1, \widehat{\mathcal{B}_1}, \epsilon)$ illustré sur la figure 5-5. Considérons maintenant la fonction $f_2(x_1, x_2) = (x_1 + 1)^2 + (x_2 + 1)^2$. L'ensemble $\mathcal{C}_2 \subset \widehat{\mathcal{C}_1}$ tel que $f_2(\mathcal{C}_2) = \widehat{\mathcal{B}_1}$ est contenu de manière garantie dans le sous-pavage $\widehat{\mathcal{C}_2} = \mathsf{SiviaSp}(\widehat{\mathcal{C}_1}, f_2, \widehat{\mathcal{B}_1}, \epsilon)$, illustré sur la figure 5-6.* ♢

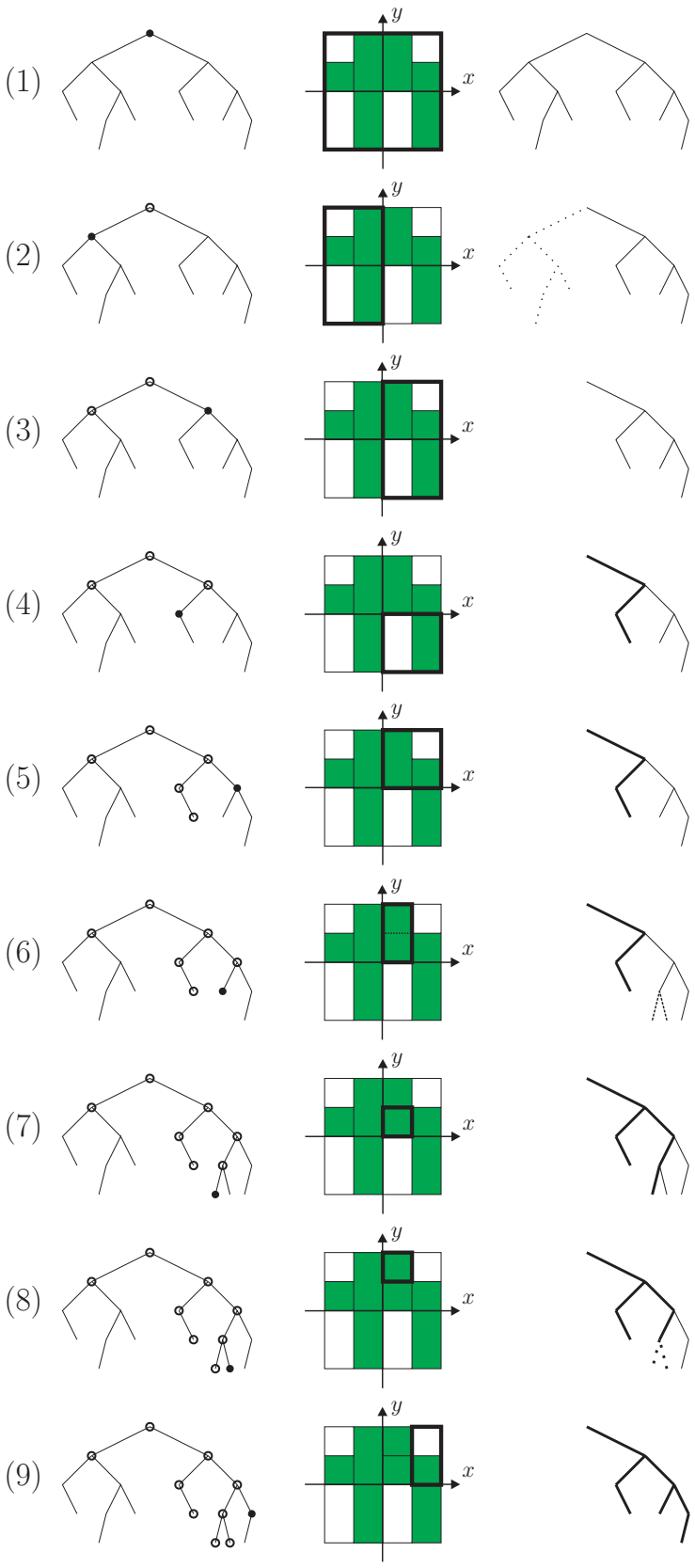


Figure 5-4: Comportement de SiviaSp *sur un sous-pavage.*

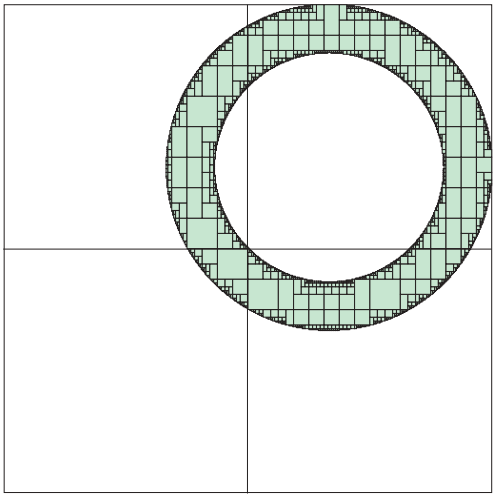


Figure 5-5: $\widehat{\mathcal{C}_1}$ calculé par SiviaSp pour l'exemple 5.4.2.

.

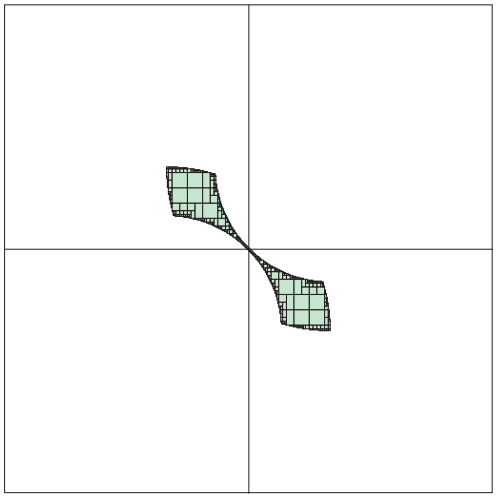


Figure 5-6: $\widehat{\mathcal{C}_2}$ calculé par SiviaSp pour l'exemple 5.4.2.

5.5 Prédiction garantie

Pour cette étape, le problème est d'évaluer une approximation extérieure de l'ensemble

$$\mathcal{X}_{k+} = \{\mathsf{f}_k(\mathsf{x}, \mathsf{u}_k, \mathsf{v}_k) \mid \mathsf{x} \in \mathcal{X}_k, \mathsf{v}_k \in [\mathsf{v}]_k\}. \tag{5.12}$$

Cette tâche peut être incluse dans le cadre plus général du problème du calcul de l'image directe d'un ensemble par une fonction, qui constitue le cœur de l'analyse par intervalles. Ce problème peut être formulé plus généralement de la manière suivante : soient deux ensembles $\mathcal{X} \subset \mathbf{R}^n$ et $\mathcal{S}_0 \subset \mathbf{R}^m$ et une fonction $\mathsf{f} : \mathbf{R}^n \to \mathbf{R}^m$, évaluer $\mathcal{S} \subset \mathcal{S}_0$ tel que $\mathcal{S} = \{\mathsf{f}(\mathsf{x}) \in \mathcal{S}_0 \mid \mathsf{x} \in \mathcal{X}\}$.

Dans le cas particulier où $m = 1$, Moore [Moore79] a montré qu'il est possible de construire une approximation aussi précise que désirée de l'évaluation intervalle d'une fonction f lorsqu'une fonction d'inclusion $f_{[]}$ lipschitz est disponible. Pour cela, il suffit de découper l'intervalle où l'on souhaite évaluer l'image en N sous-intervalles de même longueur et de calculer l'image de ces intervalles par $f_{[]}$, l'image de f sera incluse dans l'union des images précédemment calculées. Pour augmenter la précision, il suffit d'augmenter N.

Dans le cas où $m > 1$, nous allons montrer qu'un tel résultat est généralisable et qu'il est encore possible d'approcher l'ensemble image d'une fonction vectorielle f avec une précision arbitraire à l'aide d'un sous-pavage. La méthode qui sera utilisée variera suivant la disponibilité d'une expression explicite de l'inverse de f.

Pour calculer l'ensemble décrit par (5.12), lorsque f_k est inversible et lorsqu'on dispose d'une expression de f_k^{-1}, il est possible de transformer à nouveau le problème de calcul d'image directe en un problème d'inversion ensembliste ; en effet l'équation (5.12) peut se réécrire

$$\mathcal{X}_{k+} = \left\{\mathsf{x} \in \mathbf{R}^n \mid \mathsf{f}_k^{-1}(\mathsf{x}, \mathsf{u}_k) \in \mathcal{X}_k \times [\mathsf{v}]_k\right\}. \tag{5.13}$$

Cette forme est analogue à (5.8). Par conséquent, il est possible d'obtenir un sous-pavage extérieur approchant $\mathcal{X}_{k+}$ à l'aide de SiviaSp. Pour réduire l'espace de recherche de l'image directe, on peut se limiter à un sous-ensemble de $\mathbf{R}^n$. On peut, par exemple considérer le pavé $[\mathsf{s}]^0$, image par la fonction d'inclusion $\mathsf{f}_{k[]}$ de f_k du plus petit pavé contenant le sous-pavage dont il faut calculer l'image. Si on note $\mathcal{X}[\mathsf{v}]_k$ le sous-pavage contenant $\mathcal{X}_k \times [\mathsf{v}]_k$, alors une image approchée de $\mathcal{X}_{k+}$ est obtenue en évaluant

$$\widehat{\mathcal{X}_{k+}} = \mathsf{Sivia}([\mathsf{s}]^0,\ \mathsf{f}_{k[]}^{-1},\ \mathcal{X}[\mathsf{v}]_k,\ \epsilon). \tag{5.14}$$

Supposer f_k inversible peut sembler être une hypothèse relativement forte, cependant pour de nombreux systèmes physiques (par exemple les systèmes mécaniques sans frottement sec), l'inverse de la dynamique est aisée à obtenir car elle correspond à une simple inversion du temps.

Lorsque f n'est pas inversible, une procédure spécifique est nécessaire. L'idée de base de ImageSp, procédure de calcul de l'Image d'un Sous-pavage, est de décrire l'ensemble $\mathcal{X}$ dont il faut évaluer l'image à l'aide d'un sous-pavage non minimal constitué de p pavés $[\mathsf{x}]_i$ dont la plus grande longueur est inférieure à une précision spécifiée ϵ (voir figure 5-7(b)). Ensuite, l'image de ces p pavés est évaluée à l'aide d'une fonction d'inclusion $\mathsf{f}_{[]}$ de f, et ces images sont rangées dans une liste $\mathcal{L}_\epsilon$. Ainsi, p pavés *images* sont obtenus, chacun contenant l'image réelle de $[\mathsf{x}]_i$ par la fonction f. L'ensemble image $\mathcal{S}$ est de ce fait inclus

dans l'union de ces images. Enfin, ImageSp rassemble ces pavés images au sein d'un sous-pavage minimal afin de faciliter les calculs ultérieurs (voir figure 5-7(c)).

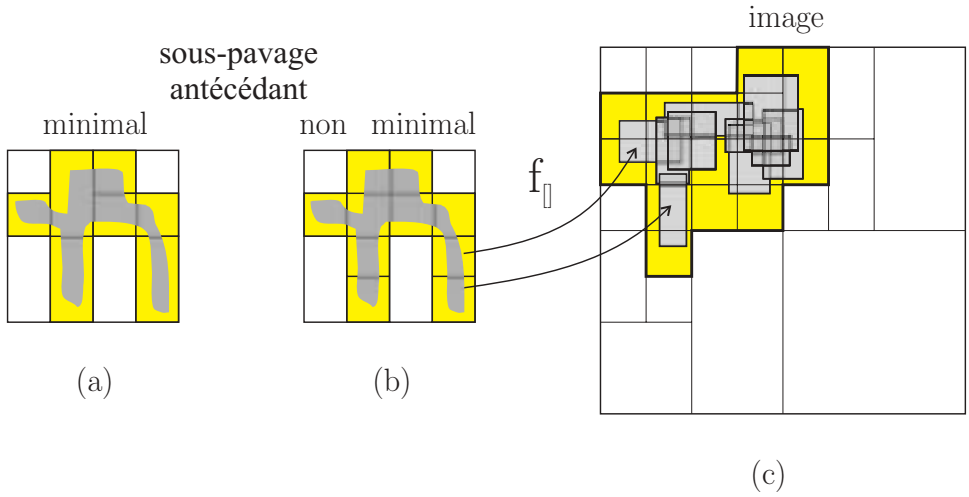


Figure 5-7: Comportement d'ImageSp.

L'implantation complète d'ImageSp peut être trouvée à la section 5.7.4. $\widehat{\mathcal{S}}_\epsilon$ =ImageSp($[s]_0$,$f_{[]}$, $\widehat{\mathcal{X}}$, ϵ) renvoie le sous-pavage $\widehat{\mathcal{S}}_\epsilon$ image approchée du sous-pavage $\widehat{\mathcal{X}}$ par la fonction $f_{[]}$, en fait une approximation de la partie de l'image incluse dans le pavé de recherche $[s]_0$.

La qualité de l'image ainsi estimée dépend de la qualité de la fonction d'inclusion $f_{[]}$ de f et du paramètre de précision ϵ. Nous allons établir qu'il est possible, au moins en principe, d'obtenir une image arbitrairement proche de l'ensemble image réel.

Proposition 6 *Soient un sous-pavage* $\widehat{\mathcal{X}} \subset \mathbf{R}^n$, *une fonction* $f : \mathbf{R}^n \to \mathbf{R}^q$ *disposant d'une fonction d'inclusion* $f_{[]}$ *lipschitz sur* $\widehat{\mathcal{X}}$, *et* $[s]^0 \subset \mathbf{R}^q$ *tel que* $f\left(\widehat{\mathcal{X}}\right) \subset [s]^0$. *Pour tout* $\eta > 0$, *il existe* $\epsilon > 0$ *tel que* $d\left(\text{ImageSp}([s]^0, f, \widehat{\mathcal{X}}, \epsilon), f\left(\widehat{\mathcal{X}}\right)\right) \leq \eta$.

Preuve : voir l'annexe D.

De la même manière qu'une fonction d'inclusion fournit un pavé encadrant l'image d'une fonction sur un pavé donné. Lorsqu'on considère un sous-pavage $\widehat{\mathcal{X}}$ donné et une fonction f disposant d'une fonction d'inclusion Lipschitz, grâce à ImageSp, il est possible d'obtenir un sous-pavage $\widehat{\mathcal{S}}_\epsilon$ contenant de manière garantie l'image de $\widehat{\mathcal{X}}$ par f. ImageSp permet donc de construire pour les sous-pavages des *fonctions d'inclusion* ayant pour valeur des *sous-pavages*.

5.6 Estimation d'état garantie

5.6.1 Algorithme approché garanti

A partir des deux étapes vues dans les paragraphes 5.4 et 5.5, il est maintenant possible de donner une version *approchée* mais *garantie* de l'algorithme 2. L'estimateur est approché car les ensembles contenant les valeurs possibles de l'état à chaque instant sont approximés par des sous-pavages ; comme cette approximation est extérieure, aucune valeur possible de l'état ne sera omise, l'estimateur est en ce sens garanti.

Algorithme 3 *Pour $l = 0$ à L, effectuer*

1. *une prédiction garantie ;* on peut utiliser

$$\widehat{\mathcal{X}_{l+}} = \mathsf{Sivia}(\widehat{\mathcal{S}},\, \mathsf{f}_{l[]}^{-1},\, \widehat{\mathcal{X}_l} \times [\mathsf{v}]_l\,,\, \epsilon);$$

   si f_l est inversible, avec $\widehat{\mathcal{S}} = \{[\mathsf{s}]\}$ un sous-pavage de recherche suffisamment grand pour contenir tous les états, ou bien, dans tous les cas,

$$\widehat{\mathcal{X}_{l+}} = \mathsf{ImageSp}(\,[\mathsf{s}]\,,\, \mathsf{f}_{l[]},\, \widehat{\mathcal{X}_l} \times [\mathsf{v}]_l\,,\, \epsilon);$$

2. *une correction garantie :*

$$\widehat{\mathcal{X}_{l+1}} = \mathsf{Sivia}(\widehat{\mathcal{X}_{l+}},\ \mathsf{h}_{l+1[]},\, \mathcal{Y}_{l+1},\, \epsilon);$$

Cet algorithme fait apparaître un paramètre ϵ qui sera déterminant sur la précision de description et les temps de calcul, comme nous allons le voir dans la suite de l'exemple présenté au paragraphe 5.2.

5.6.2 Exemple de la balle au rebond (suite)

L'algorithme 3 est appliqué à l'exemple de la balle au rebond. L'intervalle d'échantillonnage est $T = 1$ s, $r = 0.2$ m et g a été pris à 10 m.s^{-2}. L'état initial (inconnu) est $\mathsf{x}_0 = (x_0, \dot{x}_0)^{\mathrm{T}} = \left(5.2\text{ m}, 0\text{ m.s}^{-1}\right)^{\mathrm{T}}$; il se trouve dans le pavé initial $[\mathsf{x}_0] = [3\text{ m}, 6\text{ m}] \times \left[-3\text{ m.s}^{-1}, 3\text{ m.s}^{-1}\right]$. Au cours du mouvement, on suppose que l'état de la balle reste dans $[0\text{ m}, 8\text{ m}] \times \left[-12\text{ m.s}^{-1}, 12\text{ m.s}^{-1}\right]$. A chaque instant d'échantillonnage $t = kT$, $k \in \mathbf{Z}_+^*$, la hauteur de la balle est mesurée et l'erreur de mesure w_k appartient à l'intervalle $[-0.2\text{ m}, 0.2\text{ m}]$. L'équation d'observation est $y_k = x_k + w_k$. Le bruit d'état est considéré comme négligeable. On utilisera une bissection suivant la longueur maximale pondérée, avec pour poids $\mathsf{p} = \left(1\text{ m}^{-1}, 1\text{ m}^{-1}\text{s}\right)$. Le facteur de précision ϵ est de 0.1.

La fonction évaluant l'état de la balle à l'instant $(k+1)\,T$ à partir de sa position à l'instant kT fait apparaître des branchements conditionnels de type (`if...then...else...`) ; il est cependant parfaite-

Figure 5-8: Evolution de l'état de la balle, prédictions calculées par SiviaSp.

ment possible d'en obtenir une fonction d'inclusion à l'aide de la fonction χ multivaluée présentée au paragraphe 2.7. La fonction d'inclusion obtenue est donnée dans l'annexe E.2.

A chaque pas l, le sous-pavage contenant l'état prédit $\widehat{\mathcal{X}_l^+}$ est calculé grâce à SiviaSp, et est représenté sur la figure 5-8 en gris clair. Le sous-pavage corrigé $\mathcal{X}_{l+1}$ apparaît en gris foncé.

Reprenons le même exemple, mais cette fois, pour la phase de prédiction, la procédure ImageSp sera mise en œuvre. Les sous-pavages obtenus sont représentés sur la figure 5-9.

Figure 5-9: Evolution de l'état de la balle, prédictions calculées par ImageSp.

Pour comparer l'efficacité des deux algorithmes de calcul d'image directe, retenons d'une part le temps de calcul t_l nécessaire à l'évaluation de l'image d'un sous-pavage et d'autre part le coefficient d'augmentation de volume, c'est-à-dire le rapport γ_l des volumes des sous-pavages image et antécédent pour un pas donné l de l'algorithme, ce qui permet de comparer le pessimisme introduit par chacune des procédures de calcul d'image directe :

$$\gamma_l = \frac{volume\left(\widehat{\mathcal{X}_l^+}\right)}{volume\left(\widehat{\mathcal{X}_l}\right)} \tag{5.15}$$

Nous obtenons le tableau 5.1.

Dans cet exemple, pour la précision ϵ choisie, l'algorithme SiviaSp introduit un pessimisme en général

	SiviaSp		ImageSp	
l	t_l (ms)	γ_l	t_l (ms)	γ_l
1	686	1.20	245	1.33
2	501	1.34	204	1.55
3	477	1.40	158	1.51
⋮	⋮	⋮	⋮	⋮
17	151	1.77	76	1.82
18	168	1.61	62	1.74
19	189	1.58	82	1.82

Tableau 5.1: Comparaison de SiviaSp et d'ImageSp.

moins important qu'ImageSp. Par contre, cette meilleure précision se fait au prix d'un temps de calcul plus important. Une réduction du pessimisme peut être obtenue en diminuant ϵ, mais le temps de calcul augmente en conséquence. Les tableaux 5.2 et 5.3 illustrent l'effet de la variation de ϵ sur le temps de calcul τ_l et sur le coefficient d'augmentation de volume γ_l. La comparaison a été faite pour le pas $l = 11$.

ϵ	0.5	0.25	0.1	0.05
τ_l (ms)	270	370	464	762
γ_l	1.63	1.45	1.27	1.18

Tableau 5.2: Influence de ϵ sur τ_l et γ_l pour SiviaSp.

ϵ	0.5	0.25	0.1	0.05
τ_l (ms)	29	52	141	161
γ_l	1.65	1.53	1.31	1.19

Tableau 5.3: Influence de ϵ sur τ_l et γ_l pour ImageSp.

En pratique, lorsque peu d'observations sont disponibles, il convient d'utiliser l'algorithme le moins pessimiste, même s'il est un peu plus lent ; par contre, si beaucoup d'observations sont disponibles et permettent de corriger l'état prédit, peu importe d'avoir un sous-pavage image approché plus grand que l'image exacte.

5.7 Mise en œuvre des sous-pavages

Dans ce paragraphe, nous allons illustrer la représentation des sous-pavages à l'aide d'arbres binaires en présentant une partie des sources qui la mettent en œuvre. Certaines fonctions de base sur les sous-pavages ainsi que les algorithmes utilisés pour réaliser une estimation d'état garantie seront également présentés. La structure d'arbre binaire est très classique en informatique, et peut être trouvée dans la bibliothèque standard <tree.h> disponible dans tout compilateur C++. Les arbres binaires sont dans la plupart des cas destinés à construire des algorithmes de tris. Cette représentation est également utile en imagerie où on lui préfère cependant les *quadtrees* et les *octrees*, arbres à 4 ou 8 enfants, mieux adaptés à des représentation d'images en 2D ou en 3D [Klinger76]. Malheureusement, les outils de manipulation d'arbres binaires disponibles dans les bibliothèques standards ne sont absolument pas appropriés à la manipulation

de sous-pavages. Pour cette raison, nous avons préféré développer une classe `spaving` originale et des fonctions de manipulation de sous-pavages par manipulation de cette classe.

5.7.1 Classe de base spaving

Certaines indications ont été introduites au paragraphe 5.3 sur la manière dont est décrit un sous-pavage. La structure de base du sous-pavage est le nœud, correspondant à la classe `node`. Les données seront stockées dans les champs privés de cette classe. Un arbre (sous-pavage) sera un pointeur sur un nœud ; on définit donc le type `spaving` (sous-pavage) comme étant un pointeur sur un nœud

```
class node; //en-tête vide pour définir le type
typedef node* spaving;   //nouveau type spaving
```

Il faudra prêter attention à toujours initialiser le sous-pavage lors de l'emploi d'une variable de type `spaving`. On pourra par exemple l'initialiser à `NULL`, pour un sous-pavage vide. L'en-tête de la classe `node` est donné dans le tableau 5.4.

```
class node
{
     ivector* pbox;    // pavé correspondant au noeud
     spaving plchild;            // sous-arbre gauche
     spaving prchild;             // sous-arbre droit
     bool temp; // drapeau indiquant noeud temporaire
public:
//Constructeurs
     node();                                //standard
     node(ivector&);                      //initialisé
     node(node&);                        //par recopie
//Destructeur
     ~node();
//Fonctions membres
     friend ivector _box(spaving a)
          { return(*(a->pbox));  };
     friend bool isempty(const spaving);
     friend bool isleaf(const node&);
     friend void operator=(node&);
     friend spaving union(spaving,spaving,ivector&);
     ...
}
```

Tableau 5.4: Définition de la classe `node`

`pbox` est un pointeur vers le pavé représenté par le noeud courant. Le pointeur `plchild` (resp. `prchild`) désigne le sous-arbre gauche (resp. droit) issu du nœud courant. Un pointeur valant `NULL` correspond à un sous-arbre vide ; de ce fait, une feuille peut être identifiée par deux pointeurs `plchild` et `prchild` valant `NULL`. Enfin, le booléen `temp` indique si le nœud courant est temporaire ; son rôle est analogue au booléen `temp` de la classe `ivector` (cf. p. 38).

Examinons dans un premier temps les constructeurs.

Le constructeur standard (voir le tableau 5.5) est simple ; il crée une racine vide et tous les pointeurs sont initialisés à `NULL`.

```
node::node()
{
     pbox=NULL;
     plchild=NULL;  prchild=NULL;
     temp=false;
}
```

Tableau 5.5: Constructeur standard de `node`

Le constructeur initialisé à partir d'un `ivector` (voir le tableau 5.6) ne diffère pour l'essentiel du constucteur standard que par le fait qu'un nœud réduit à un pavé racine est créé. Le pointeur sur le pavé correspondant à la racine est initialisé. Il pointe sur une copie du vecteur `v`.

```
node::node(ivector& v)
{
    pbox=new ivector(v);
    plchild=NULL;  prchild=NULL;
    temp=false;
}
```

Tableau 5.6: Constructeur initialisé de `node`

Le constructeur par recopie (tableau 5.7) donne un premier aperçu d'une structure d'algorithme de traitement récursif sur les arbres binaires. Dans une première partie, le contenu du nœud est testé et modifié, ensuite, la procédure est appliquée récursivement au sous-arbre gauche, puis au sous-arbre droit. L'ordre de traitement : nœud - sous-arbre gauche - sous-arbre droit n'est pas arbitraire et peut varier suivant l'algorithme. Traiter le contenu du nœud en dernier permet par exemple d'analyser tout d'abord les feuilles de l'arbre puis de remonter vers la racine.

Si le nœud à copier est vide, un nœud vide est créé. Si ce n'est pas le cas, le pavé correspondant au nœud `a` est copié, et le drapeau `temp` est placé à `false` car le nœud créé n'est pas résultat d'une opération. Ensuite, on teste si `a` a un fils gauche. Si c'est le cas, le pointeur vers le fils gauche du nœud courant est positionné sur une copie du sous-pavage gauche de `a` en appelant récursivement le constructeur par recopie. Evidemment le même traitement est effectué également sur le fils droit. En fin de traitement, si `a` est temporaire, il est détruit.

Le destructeur (voir tableau 5.8) présente la même structure de parcours de l'arbre que le constructeur par recopie, par contre le sens de parcours n'est pas le même.
Si un nœud est vide, rien n'est fait. Sinon, la destruction des sous-arbres gauche et droit est faite lorsqu'ils existent. Après cela, le pavé correspondant au nœud courant est détruit.

La procédure testant si un sous-pavage est vide est très simple. Comme il est en fait représenté par un pointeur, un sous-pavage est vide lorsque le pointeur vaut `NULL` (tableau 5.9).

```
node:: node(node& a)
{
// traitement du contenu du nœud
     if (isempty(a))
     {
          pbox=NULL;
          plchild=NULL;  prchild=NULL;
          temp=false;
          return;
     }
     pbox=new ivector(*(a.pbox));
     temp=false;
// traitement du fils gauche
     if (isempty(a.plchild)) plchild=NULL;
     else plchild=new node(*(a.plchild));
// traitement du fils droit
     if (isempty(a.prchild)) prchild=NULL;
     else prchild=new node(*(a.prchild));

     if (a.temp) delete (&a);
}
```

Tableau 5.7: Constructeur par recopie de `node`

```
node:: ~node()
{
     if (isempty(this)) return;
     if (plchild) delete plchild;
     if (prchild) delete prchild;
     delete pbox;
}
```

Tableau 5.8: Destructeur de `node`

```
bool isempty(const spaving spavage)
{
   return (spavage==NULL);
}
```

Tableau 5.9: `isempty` pour un `spaving`

```
bool isleaf(const node& noeud)
{
     return (isempty(noeud.plchild)&&isempty(noeud.prchild));
}
```

Tableau 5.10: `isleaf` pour un `node`

Tester si un nœud est une feuille se fait en vérifiant qu'il n'a ni fils droit, ni fils gauche (tableau 5.10).

Une fonction utile pour la manipulation de sous-pavages, est celle réalisant la réunion de deux sous-pavages frères afin d'obtenir le sous-pavage père minimal (cf. p. 5.4.1). Cette fonction est présentée dans le tableau 5.11.

```
spaving reunion(spaving pl, spaving pr, ivector& pavepere)
{
     if (isempty(pl)&&isempty(pr)) return NULL;
// Création du noeud père commun
     spaving pres=new node(pavepere);
     pres->temp=true;       // le résultat est temporaire
// Rajoût des sous-arbres sauf si ce sont des feuilles
     if (isempty(pl))
     {    pres->plchild=NULL;  pres->prchild=pr;  }
     else if (isempty(pr))
     {    pres->plchild=pl;  pres->prchild=NULL;  }
     else if (isleaf(*pl)&&isleaf(*pr))
     {
          if (pl->temp) delete pl;
          if (pr->temp) delete pr;
     }
     else
     {    pres->plchild=pl;  pres->prchild=pr;  }

     return (pres);
}
```

Tableau 5.11: `reunion` de deux `spaving`

Le comportement de cet algorithme dans diverses situations est illustré par la figure 5-10. Sur les arbres des deux colonnes de gauche, les points noirs désignent les nœuds pointés repectivement par `pl` et `pr` ; à droite, ils désignent le nœud-père pointé par `pres`. Lorsque deux sous-pavages vides sont transmis, la réunion des deux est vide. Dans le cas contraire, le nœud père des deux frères est construit à partir du constructeur initialisé par le pavé `pavepere`. Ce nœud est temporaire car il résulte d'une opération entre sous-pavages. Lorsque les deux fils sont des feuilles, ils sont détruits s'ils sont temporaires (voir la figure 5-10 (i)). Dans le cas contraire, les pointeurs vers le fils droit et vers le fils gauche du pavé-père sont ajustés (voir la figure 5-10 (ii) et (iii)).

5.7.2 Autres fonctions

SiviaSp et ImageSp mettent en œuvre un certain nombre de fonctions secondaires qui sont présentées dans ce paragraphe.

Découpage d'un sous-pavage

Le fonctions de découpage des sous-pavages sont essentielles dans les procédures récursives. Par exemple, lorsque le résultat de SiviaSp est indéterminé sur une feuille, le pavé correspondant doit être bissecté, la

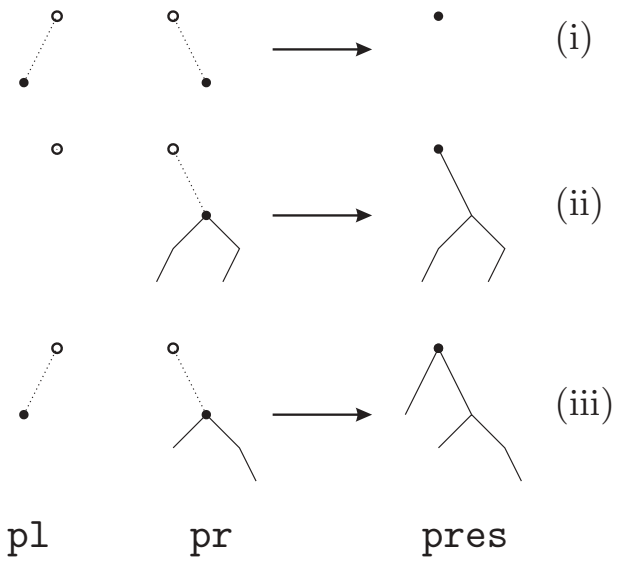


Figure 5-10: Réunion de deux sous-pavages frères.

feuille devient donc un nœud d'où partent deux sous-arbres qu'il faut examiner récursivement.

Les algorithmes `expanddepth` et `expandwidth` correspondent à des implantations qui ne posent pas de difficulté algorithmique, ils ne seront pas détaillées. Seuls leur prototype et leur comportement seront explicités.

La fontion

```
void expanddepth(spaving spav,int depth,bissectype cut,int bisecdir=1);
```

permet de modifier `spav` de manière à obtenir un sous-pavage non minimal dont toutes les feuilles sont au moins à la profondeur `depth`. La bissection des pavés se fait selon la loi spécifiée dans `cut` ; les pavés sont coupés dans leur dimension soit de plus grande longueur (`cut=_maxlong`), soit de plus grande longueur relative (`cut=_maxrellong`), soit en incrémentant la direction de coupe à chaque appel (`cut=_altern`). Dans ce dernier cas, la première direction de coupe `bisecdir` doit être spécifiée.

Exemple 5.7.1 *Considérons par exemple un sous-pavage* `spav` *constitué d'un pavé unique de dimension* 4. *bissecte le pavé correspondant à* `spav` 5 *fois, d'abord suivant sa dimension* 3, *ensuite suivant* 4, *puis*

```
expanddepth(spav,5,_altern,3);
```

1, *puis* 2 *et enfin suivant la dimension* 3. *Le sous-pavage* `spav` *aura une profondeur de* 5. ♢

La fontion

```
void expandwidth(spaving spav,float eps,bissectype cut,int bisecdir=1);
```

permet de modifier `spav` de manière à obtenir un sous-pavage non minimal dont toutes les feuilles ont une longueur maximale (si `cut` vaut `_altern` ou `_maxlong`) ou une longueur relative maximale (si `cut`

vaut _maxrellong) inférieure à eps. La bissection des pavés utilise les paramètres cut et bisecdir de la même manière que expanddepth.

Inclusion d'un pavé dans un sous-pavage

Déterminer si un point ou un pavé appartient à un ensemble donné est en général une tâche assez difficile. Grâce à la notion de sous-pavage et à leur description récursive sous forme d'arbre binaires, il est possible de construire un test récursif très simple apportant une solution à ce problème.

```
bool inclus (ivector& v, spaving spav)
{
     if isempty(v)
          return true;
     if isempty(spav)
          return false; // v n'est pas vide et spav l'est
     if isleaf(*spav)
          return (inclus(v,_box(spav))
// récursivité
     ivector lbox(_box(spav)),rbox(_box(spav));
     Bissect(_box(spav),lbox,rbox,_maxlong);

     return (inclus(v&lbox,spav->plchild)
                         &&inclus(v&rbox,spav->prchild));
}
```

Tableau 5.12: Test d'appartenance d'un pav'e à un sous-pavage.

La fonction inclus avec pour arguments un ivector et un spaving détermine si un pavé v est inclus dans un sous-pavage spav. Si le pavé v est vide, alors il est inclus dans le sous-pavage. S'il n'est pas vide, mais si par contre spav l'est, alors v ne peut être inclus dans le sous-pavage. Lorsque le nœud pointé par spav est en fait une feuille, le sous-pavage est réduit au pavé _box(spav), et il suffit de tester si v est inclus dans ce pavé.

Lorsqu'aucune de ces conditions n'est satisfaite, on considère les sous-pavés lbox et rbox correspondant respectivement aux fils gauche et droit du nœud pointé par spav. Ensuite, on réalise une partition de v : une partie correspondant au sous-pavage gauche et une partie correspondant au sous-pavage droit. Le test est appliqué récursivement à l'intersection (opérateur &) de v avec lbox et au sous-arbre gauche spav->plchild et à l'intersection de v avec rbox et au sous-arbre droit spav->prchild. Pour que le pavé v soit inclus dans le sous-pavage, il faut que ses deux partie le soient.

La figure 5-11 illustre le comportement de cet algorithme sur un exemple. Le pavé à tester est entouré de pointillés, le sous-pavage est grisé. Au pas (1), rien ne peut être décidé. Le pavé est réparti entre le sous-arbre gauche et le sous-arbre droit. Au pas (2), on voit que la partie de v correspondant au sous-arbre gauche est vide, le test est donc vrai. Pour la partie correspondant au sous-arbre droit, un nouvel appel récursif est nécessaire. Au pas (3), l'algorithme arrive sur une feuille dans laquelle la partie correspondante de v est inclus, pour la partie restante, un nouvel appel récursif est effectué. Au pas (4), une partie de v

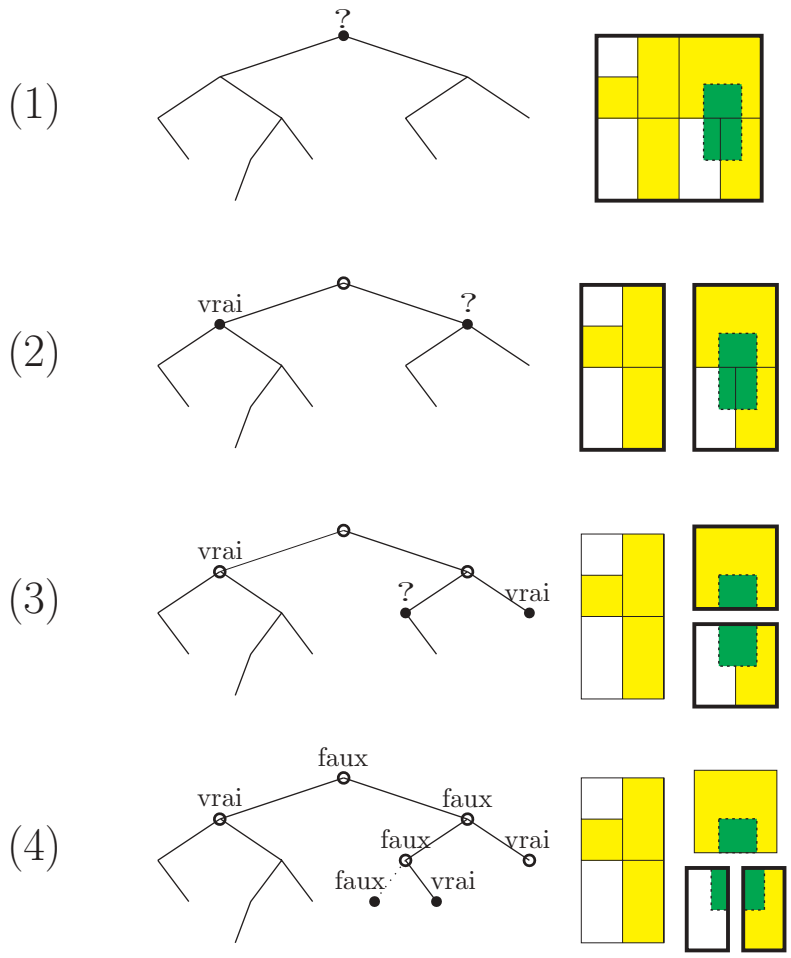


Figure 5-11: Test d'inclusion d'un pavé (en pointillés) dans un sous-pavage (en gris clair).

correspond à un sous-pavage vide, v ne peut donc pas être inclus dans le sous-pavage initial.

5.7.3 Implantation de SiviaSp

```
spaving SiviaSp(spaving pa, boolintervalFctPtr f, double eps, void* pargs)
{
     if (isempty(pa)) return NULL;  //noeud vide, renvoi d'un spaving vide
     boolinterval test=f(_box(pa),pargs);

     if (test==FAUX)                //test faux, renvoi d'un spaving vide
          return NULL;

     if (test==VRAI)          //test vrai, renvoi d'un spaving copie de pa
     {
          spaving pres=new node(*pa);  pres->temp=true;
          return (pres);
     }

     if (MaxDiam(_box(pa))<=eps)            //précision maximale atteinte
     {              //renvoi d'une feuille correspondant au pavé en cours
          spaving pres=new node(_box(pa));  pres->temp=true;
          return (pres);
     }

// Traitement récursif
     if (isleaf(*pa)) expanddepth(pa,1,_maxlong);
     return (reunion(SiviaSp(pa->lchild,f,eps,pargs),
                              SiviaSp(pa->rchild,f,eps,pargs),_box(pa)));
}
```

Tableau 5.13: SiviaSp

La procédure principale de SiviaSp est décrite dans le tableau 5.13. Si le sous-pavage de recherche pa est vide, un sous-pavage vide est renvoyé. La variable boolinterval test contient le résultat de la fonction de test f sur le pavé _box(pa) correspondant au nœud en cours d'analyse. Le pointeur de type void permet de passer tout type d'argument à la fonction de test qui ainsi peut prendre de très nombreuses formes (voir l'exemple 5.7.2). Si le test est faux, un sous-pavage vide est renvoyé. Si le test est vrai, une copie de l'arbre issu du nœud en cours est renvoyée. Enfin, si la précision maximale de recherche a été atteinte, une feuille correspondant au nœud en cours est renvoyée. Lorsqu'aucune de ces trois conditions n'est remplie, on fait croître le nœud courant si c'est une feuille de taille suffisante et SiviaSp est appliqué récursivement sur les deux feuilles ainsi créées.

Exemple 5.7.2 *Soit la fonction* $f(x_1, x_2) = x_1^2 + x_2^2$. *On cherche à caractériser*

$$\mathcal{S} = \{(x_1, x_2) \in [-2, 2] \times [-2, 2] \,|\, f(x_1, x_2) \in [0.5, 1]\}.$$

Ce problème d'inversion ensembliste peut être résolu de manière approchée par SiviaSp*. Le programme correspondant est donné dans le tableau 5.14. Un sous-pavage approchant la solution est construit par* SiviaSp *et stocké dans* solution*.* ♢

```
// Fonction de test
boolinterval fcarre(ivector& x, void* pargs)
{
     interval img=sqr(x[1])+sqr(x[2]);
//image incluse dans l'intervalle à inverser ?
     return (inclus(img,(interval*)pargs));
}

// Programme principal
void main()
{
// Initialisation
     interval a_inverser(0.5,1);
     ivector searchv(2);
     searchv[1]=_interval(-2,2);
     searchv[2]=_interval(-2,2);
     spaving searchsp=new node(searchv);
// Résolution du problème
     spaving solution=SiviaSp(searchsp,fcarre,0.01,&a_inverser);
// Affichage, etc.
     ...
}
```

Tableau 5.14: Exemple d'utilisation de SiviaSp

Il est important de noter que le programme ci-dessus est l'implantation *réelle* de SiviaSp. En quinze lignes de code, il est donc possible de réaliser un algorithme complet d'inversion ensembliste. L'introduction de la notion de sous-pavage peut paraître assez lourde, mais la concision des algorithmes récursif réalisés à l'aide de cette notion justifie pleinement à nos yeux l'investissement de départ. L'algorithme présenté réalise une bissection suivant le diamètre maximal. Pour réaliser d'autres types de bissections, il conviendrait de modifier légèrement l'algorithme, mais sa structure resterait globalement identique.

5.7.4 Implantation de ImageSp

L'algorithme ImageSp ne sera pas détaillé. En particulier, les procédures d'initialisation et de bissection ne seront pas présentées, pour ne pas surcharger l'exposé. Leur structure est dans la plupart des cas relativement simple. La structure de base d'ImageSp est présentée dans le tableau 5.15.

Dans un premier temps, le sous-pavage antécédant ante est transformé par expandwidth en un sous-pavage non minimal constitué de pavés dont la plus grande longueur est inférieure à eps. On bissecte les pavés dans leur dimension de plus grande longueur. Les images de ces pavés par la fonction d'inclusion vectorielle f sont placées dans la liste list par la fonction FillList. La procédure BuildSp est ensuite appelée pour rassembler toutes ces images au sein d'un sous-pavage inclu dans le pavé de recherche imbox

```
spaving ImageSp(ivector_FctPtr f,spaving ante,ivector imbox,double eps)
{
     PairPtr list=EmptyList;
// Modification du sous-pavage et remplissage de la liste image list
     expandwidth(ante,eps,_maxlong);
     FillList(list,f,ante);
// Procédure de construction du sous-pavage image à partir de list
     return BuildSp(list,imbox,eps);
}
```

Tableau 5.15: `ImageSp`

(éventuellement très grand) qui constituera la racine du sous-pavage image (voir la figure 5-7).

L'idée de base de `BuildSp` (voir le tableau 5.16) est analogue à celle des algorithmes *Split and Merge* (couper puis rassembler) utilisés en segmentation d'images (voir par exemple [Klinger76]).

```
spaving BuildSp(PairPtr list,ivector& imbox,double eps)
{
     if (list==EmptyList) return NULL;

     if (inlist(imbox,list))||(MaxDiam(imbox)<eps))
     {
          spaving pres=new node(imbox);  pres->temp=true;
          return pres;
     }
// Traitement récursif
     ivector limbox,limbox;
     PairPtr llist=EmptyList;  PairPtr rlist=EmptyList;
     Bissect(imbox,limbox,limbox,_maxlong);
     Distribute(list,imbox,llist,rlist);
     return reunion(BuildSp(llist,limbox,eps),
                         BuildSp(rlist,rimbox,eps),imbox);
}
```

Tableau 5.16: `BuildSp`

Si la liste `list` est vide, un sous-pavage vide est renvoyé. Sinon, `BuildSp` teste si le pavé courant `imbox` est contenu dans la liste, ce qui signifie que le pavé appartient à l'ensemble image. En outre, si la longueur maximale de `imbox` est plus petite que `eps`, il doit également être stocké dans le sous-pavage image approchée car l'algorithme a échoué dans sa tentative de prouver que `imbox` et l'ensemble image sont disjoints. Si aucune de ces deux condition n'est satisfaite, le pavé *a priori* `imbox` est coupé en `limbox` et en `rimbox`, la liste `List` est également divisée par la procédure `distribute` en deux sous-listes `llist` et `rlist` contenant les intersections des éléments de `list` avec respectivement `limbox` et `rimbox`. La procédure `BuildSp` est appelée récursivement, avec pour arguments les sous-listes et les sous-pavés créés. Enfin, `BuildSp` retourne le sous-pavage correspondant à l'union des deux sous-pavages renvoyés par les deux appels récursifs de `BuildSp`.

5.8 Conclusion

Dans ce chapitre, un nouvel algorithme récursif d'estimation d'état non linéaire a été présenté, capable de fournir à chaque instant d'échantillonnage un ensemble contenant de manière garantie toutes les valeurs de l'état compatibles avec les informations disponibles. Comme le filtre de Kalman classique, cet estimateur alterne des phases de prédiction et de correction.

L'étape de *prédiction* utilise soit un algorithme d'évaluation directe d'image (ImageSp), soit une procédure équivalente évaluant l'antécédant de la fonction réciproque exploitant l'algorithme SiviaSp. ImageSp coupe les pavés constituant le sous-pavage antécédant en pavés plus petits et rassemble leurs images en un sous-pavage image approchée. SiviaSp réalise une opération dans la direction opposée ; partant de l'ensemble image, les pavés ayant pu être l'image d'une partie de l'antécédant sont recherchés. Les avantages et inconvénients ces deux méthodes, ainsi que leurs propriétés de convergence ont été présentés.

L'étape de *correction* sélectionne dans l'ensemble calculé à l'étape de prédiction précédente tous les états compatibles avec les observations nouvellement acquises.

L'estimateur résultant de l'enchainement cyclique de ces deux étapes a été appliqué à un exemple simple mais représentatif de difficultés rencontrées dans l'estimation d'état de systèmes non linéaires ou hybrides.

L'introduction des sous-pavages est un élément-clef dans l'implantation récursive de l'algorithme. Ces sous-pavages permettent de calculer des encadrements extérieurs d'ensemble avec une précision arbitraire. En faisant une surcharge des opérateurs arithmétiques standards et des fonctions usuelles sur les flottants il sera facilement possible d'obtenir *une arithmétique sur les sous-pavages* approximant l'arithmétique sur les ensembles de réels.

La complexité augmentant de manière exponentielle avec le nombre de variables est la principale limitation d'une telle technique d'estimation d'état. Cependant, il est possible de traiter de nombreux problèmes réalistes de poursuite. Au chapitre suivant, nous allons nous intéresser au problème de la poursuite d'un robot mobile.

Chapitre 6

Poursuite d'un robot mobile

6.1 Introduction

Le chapitre 5 a présenté un algorithme d'estimation d'état récursif dans le contexte erreurs bornées utilisant l'analyse par intervalles. Cet estimateur fournit à chaque instant un ensemble qui contient de manière garantie toutes les valeurs de l'état compatibles avec les informations disponibles.

L'objectif de ce chapitre est de montrer que cet algorithme d'estimation d'état peut être appliqué sans difficultés à un problème beaucoup plus réaliste que le suivi d'une balle rebondissant, à savoir la poursuite d'un robot mobile tel que celui représenté sur la figure 4-1. Nous avons vu que la localisation statique garantie d'un robot est un processus relativement long (de 10 s à 1 ou 2 mn suivant les cas), car l'espace des configurations à explorer est vaste. En tirant parti de la récursivité de l'algorithme d'estimation d'état, il est parfaitement possible, une fois cette localisation effectuée d'alterner des phases de prédiction et de correction, en disposant avant chaque correction d'un ensemble restreint de configurations potentielles pour lequel une phase de localisation est potentiellement beaucoup plus rapide.

Pour la phase de prédiction, une équation d'évolution de l'état est indispensable ; une équation d'évolution de la configuration du robot sera donc présentée au paragraphe 6.2. La phase de correction met en œuvre l'algorithme de localisation statique décrit au chapitre 4. Nous passerons par conséquent très brièvement sur cet aspect avant de présenter l'algorithme d'estimation d'état global pour le robot. Au paragraphe 6.3, différents exemples seront présentés, qui mettront en évidence la nécessité de l'étape de correction, la possibilité d'existence d'ambiguïtés dues aux symétries du problème et la possibilité de traiter des données aberrantes (voir également [Kieffer et al.99b]).

6.2 Equations d'état et d'observation

Afin d'utiliser l'algorithme d'estimation d'état pour le suivi du robot mobile, les deux fonctions $\mathbf{f}$ et $\mathbf{h}$ correspondant respectivement à l'équation d'évolution et à l'équation d'observation du système (5.1)

doivent être définies.

L'équation d'observation correspond à la fonction permettant de calculer l'espacement entre chaque capteur et la carte fournie au robot (voir le chapitre 4). Ainsi, si on note l'état du robot $\mathsf{x}_l = \left((x_c)_l, (y_c)_l, (\theta)_l\right)^{\mathrm{T}}$ au pas l, la partie déterministe du vecteur des observations est alors

$$\mathsf{y}_l = \left(r_1\left(\mathsf{x}_l\right), \ldots, r_{n_\mathrm{s}}\left(\mathsf{x}_l\right)\right)^{\mathrm{T}}, \tag{6.1}$$

où n_s représente le nombre de capteurs et r_j l'éloignement (voir le chapitre 4).

Pour calculer l'évolution de l'état du robot, nous considérerons les équations cinématiques du mouvement. Le modèle résultant n'est valable que pour des vitesses de déplacement suffisamment faibles. Les deux roues motrices, commandées indépendamment, ont même rayon R. Leur vitesse de rotation angulaire est ω_g pour la roue gauche et ω_d pour la roue droite ; elles sont supposées rouler sans glisser. Les deux autres roues sont libres. L'origine c du repère $\mathcal{R}$ est prise au milieu du segment joignant les deux roues motrices (voir la figure 6-1). La distance entre les deux roues motrices (appelée *voie*) est notée d.

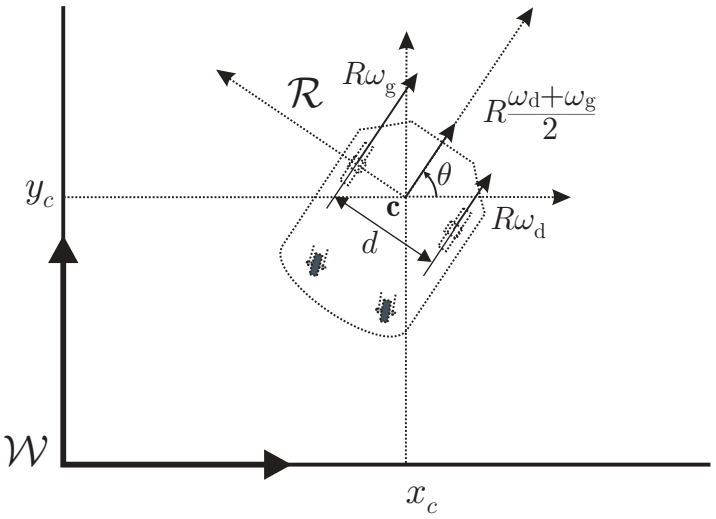


Figure 6-1: Schématisation du robot.

Dans ces conditions, l'évolution de l'état est donnée par

$$\begin{cases} \frac{\mathrm{d}x_\mathsf{c}}{\mathrm{d}t} = R\frac{\omega_\mathrm{d}+\omega_\mathrm{g}}{2}\cos\left(\theta\left(t\right)\right), \\ \frac{\mathrm{d}y_\mathsf{c}}{\mathrm{d}t} = R\frac{\omega_\mathrm{d}+\omega_\mathrm{g}}{2}\sin\left(\theta\left(t\right)\right), \\ \frac{\mathrm{d}\theta}{\mathrm{d}t} = R\frac{\omega_\mathrm{d}-\omega_\mathrm{g}}{2d}. \end{cases} \tag{6.2}$$

Une discrétisation exacte du système (6.2) est réalisée pour obtenir l'état au pas $l+1$ connaissant l'état au pas l. L'intervalle d'échantillonnage est noté T, et les vitesses angulaires de rotation des roues sont supposées constantes entre deux échantillons successifs. On peut alors aisément déduire x_{l+1} de x_l lorsque

$\omega_{\mathrm{d}} \neq \omega_{\mathrm{g}}$

$$\mathsf{x}_{l+1} = \begin{pmatrix} (x_c)_{l+1} \\ (y_c)_{l+1} \\ (\theta)_{l+1} \end{pmatrix} = \begin{pmatrix} (x_c)_l + 2d\frac{\omega_{\mathrm{d}}+\omega_{\mathrm{g}}}{\omega_{\mathrm{d}}-\omega_{\mathrm{g}}} \cos\left((\theta)_l + T\frac{R(\omega_{\mathrm{d}}-\omega_{\mathrm{g}})}{4d}\right) \sin\left(T\frac{R(\omega_{\mathrm{d}}-\omega_{\mathrm{g}})}{4d}\right) \\ (y_c)_l + 2d\frac{\omega_{\mathrm{d}}+\omega_{\mathrm{g}}}{\omega_{\mathrm{d}}-\omega_{\mathrm{g}}} \sin\left((\theta)_l + T\frac{R(\omega_{\mathrm{d}}-\omega_{\mathrm{g}})}{4d}\right) \sin\left(T\frac{R(\omega_{\mathrm{d}}-\omega_{\mathrm{g}})}{4d}\right) \\ (\theta)_l + T\frac{R(\omega_{\mathrm{d}}-\omega_{\mathrm{g}})}{2d} \text{ modulo } 2\pi \end{pmatrix}. \tag{6.3}$$

Lorsque $\omega_{\mathrm{d}} = \omega_{\mathrm{g}}$, l'expression est la suivante

$$\mathsf{x}_{l+1} = \begin{pmatrix} (x_c)_{l+1} \\ (y_c)_{l+1} \\ (\theta)_{l+1} \end{pmatrix} = \begin{pmatrix} (x_c)_l + T\frac{R(\omega_{\mathrm{d}}+\omega_{\mathrm{g}})}{2} \cos(\theta)_l \\ (y_c)_l + T\frac{R(\omega_{\mathrm{d}}+\omega_{\mathrm{g}})}{2} \sin(\theta)_l \\ (\theta)_l \end{pmatrix}. \tag{6.4}$$

Pour tenir compte d'un éventuel glissement ou d'une variation de la vitesse de rotation des roues motrices entre deux instants d'échantillonnage successifs, il suffirait d'ajouter un bruit borné à chacune des composantes de l'état, les bornes du bruit dépendant des conditions expérimentales.

Remarque 6.2.1 *Cette modélisation utilise une paramétrisation en t du mouvement. Il serait également possible d'utiliser l'abscisse curviligne pour décrire l'évolution de l'état, mais la paramétrisation en t nous a paru la plus simple à mettre en œuvre.* ♢

L'hypothèse d'une vitesse de rotation constante entre les instants d'échantillonnage est assez grossière, car elle implique des accélérations infinies à ces instants. Trois solutions peuvent être envisagées pour affiner le modèle. La première, déjà évoquée, est d'introduire un bruit d'état dans lequel serait incorporé tous les effets non modélisés. Une deuxième option consiste à considérer des vitesses affines par morceaux, ce qui traduit une accélération angulaire constante entre les instants d'échantillonnages, mais ne tient pas compte de l'inertie du robot. Enfin, la solution la plus complexe, mais la plus réaliste également, revient à considérer les couples moteurs appliqués aux deux roues. Cette troisième solution nécessite une mise en équation faisant intervenir deux variables d'état supplémentaires pour tenir compte de la vitesse de rotation des roues. Cette méthode, compte-tenu de la complexité des algorithmes Sivia et ImageSp, n'est guère réaliste au niveau des temps de calcul, du moins pour l'instant.

Pour illustrer les propriétés de l'algorithme de localisation, une modélisation assez grossière, dans laquelle ont fait éventuellement intervenir un bruit d'état, s'avère amplement suffisante.

6.3 Applications

La localisation et le suivi du robot mobile vont maintenant être illustrés sur quatre exemples simulés, mais néanmoins réalistes. Les caractéristiques du robot sont toujours celles du modèle représenté sur la figure 4-1. Il dispose de $n_{\mathrm{s}} = 24$ capteurs à ultrasons, chacun a un demi-angle d'émission $\tilde{\gamma} = 0.2$ rad et une incertitude de mesure $\alpha = 2\%$ dans son domaine de fonctionnement normal. La voie d est incertaine, elle varie dans l'intervalle [0.43 m,0.56 m], en première approximation, nous prendrons

$d = 0.5$ m. Pour tous les exemples, le domaine de recherche de l'état du robot sera $[-12\text{ m}, 12\text{ m}] \times [-12\text{ m}, 12\text{ m}] \times [0\text{ rad}, 2\pi\text{ rad}]$. SiviaSp et ImageSp utilisent une bissection des pavés à traiter suivant leur dimension de plus grande longueur pondérée, dans les différents exemples, le vecteur de poids est $\left(1\text{ m}^{-1}, 1\text{ m}^{-1}, 1\text{ rad}^{-1}\right)$ et le paramètre de précision ϵ est pris égal à 0.1. L'intervalle d'échantillonnage considéré est $T = 2$ s. On supposera que le robot est placé dans une pièce exactement décrite par la carte représentée sur la figure 6-2. Tous les calculs ont été effectués sur un Pentium 233MMX, en utilisant notre implantation en C++ des algorithmes décrits dans les chapitres 4 et 5.

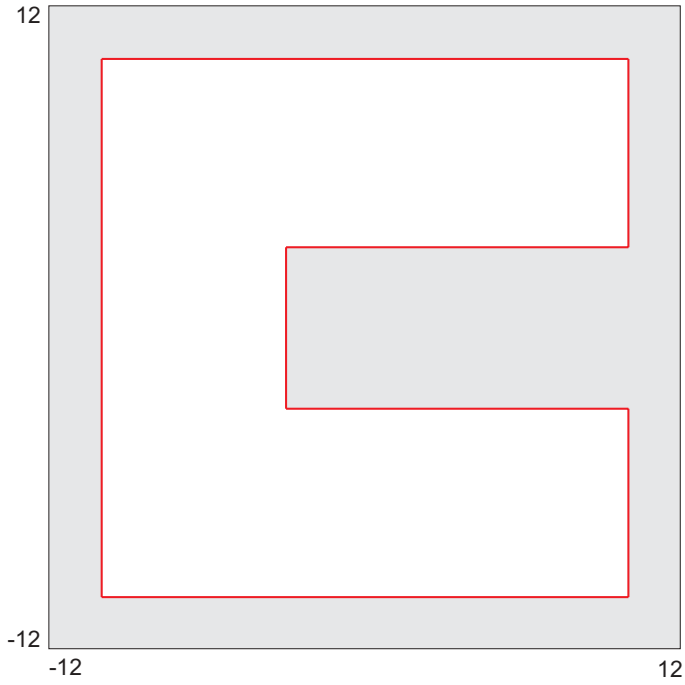


Figure 6-2: Carte utilisée par le robot.

En pratique, l'intervalle d'échantillonnage ne peut pas être pris arbitrairement petit. En effet, la mesure de distances par les capteurs à ultrasons n'est pas immédiate, d'une part à cause de la vitesse de propagation limitée des ondes sonores (environ 330 m.s^{-1}), et d'autre part à cause du fait que les mesures ne sont pas simultanées pour l'ensemble de la ceinture de capteurs. En effet, les capteurs font des mesures par groupes de 4, de manière à ne pas se perturber les uns les autres. Une série de mesures prenant au maximum 200 ms, la ceinture de 24 capteurs fournit par conséquent un ensemble complet de mesures toutes les 1.2 s.

Pour les différents tests, nous supposerons pour simplifier la simulation que la prise de mesure est instantanée et que tous les capteurs fournissent une mesure au même instant correspondant effectivement à la position du robot à *cet* instant. Une étape de correction utilisant seulement 4 capteurs sur les 24 pourrait assez facilement être envisagée : dans l'ensemble prédit, il suffirait d'éliminer les états non compatibles avec les 4 dernières mesures obtenues. La prise en compte exacte du retard de mesure est

quant à elle beaucoup plus délicate ; une solution simple à ce problème consisterait à introduire un bruit d'état pour tenir compte du déplacement du robot (de l'ordre de quelques centimètres) pendant la mesure.

6.3.1 Premier exemple : suivi sans correction

Ce premier exemple est réalisé pour illustrer les propriétés de l'étape de prédiction. L'étape de correction ne sera utilisée qu'à l'initialisation ; ensuite, seule l'étape de prédiction sera appliquée, et aucune correction de l'état prédit ne sera effectuée. Ainsi, l'état du robot au pas l ne sera déduit que de la connaissance approximative de l'état initial et de l étapes de prédiction.

Le diagramme d'émission des 24 capteurs lors de la phase de localisation statique est représenté sur la figure 6-3. Rappelons que deux arcs sont associés à chaque capteur. Un segment de la carte doit au moins en partie se trouver entre chacun de ces arcs.

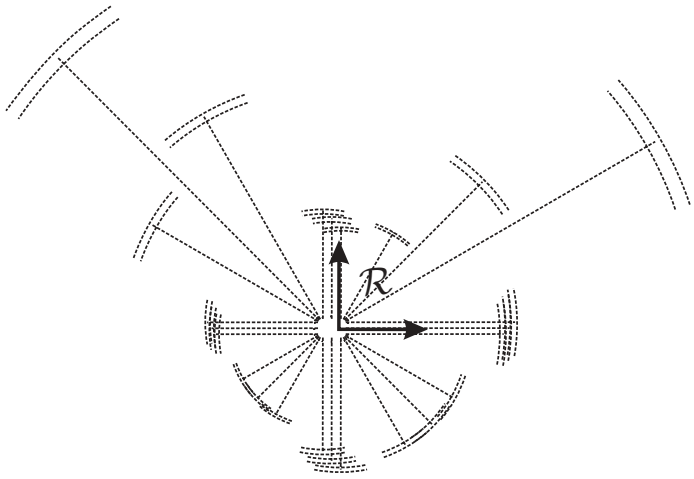


Figure 6-3: Diagramme d'émission lors de l'initialisation (premier exemple) ; l'échelle est la même que sur la figure 6-2.

Ce diagramme est obtenu grâce à l'algorithme de calcul de l'éloignement présenté au paragraphe 4.3 pour l'état initial réel $(x_c, y_c, \theta)_0 = (-5 \text{ m}, 6 \text{ m}, 4 \text{ rad})$. Cette information n'est évidemment pas mise à disposition de l'algorithme de suivi d'état. Tout d'abord, une localisation statique est opérée. Cette étape correspond à une étape de correction avec pour espace des configurations admissibles *a priori* tout l'espace de recherche *a priori*. Un premier sous-pavage, inclus dans le pavé $[-5.09 \text{ m}, -4.89 \text{ m}] \times [5.93 \text{ m}, 6.07 \text{ m}] \times [3.99 \text{ rad}, 4.01 \text{ rad}]$, et contenant de manière garantie l'état à l'instant initial, est obtenu en moins de 8 s. Le robot se déplace ensuite ; la roue gauche a une vitesse linéaire $v_\text{g} = R\omega_\text{g} = 0.65 \text{ m.s}^{-1}$ et la roue droite a une vitesse $v_\text{d} = R\omega_\text{d} = 0.75 \text{ m.s}^{-1}$. La trajectoire du robot doit ainsi être un arc de cercle dans le plan (x, y). L'étape de prédiction est appliquée pour $lT \in [0 \text{ s}, 12 \text{ s}]$. A chaque pas l, le sous-pavage prédit $\widehat{\mathcal{X}_l^+}$ est évalué par l'algorithme ImageSp, et est représenté sur les figures 6-4 et 6-5. Aucune étape de correction n'est effectuée, de ce fait, $\mathcal{X}_{l+1}$ est pris égal à $\widehat{\mathcal{X}_l^+}$.

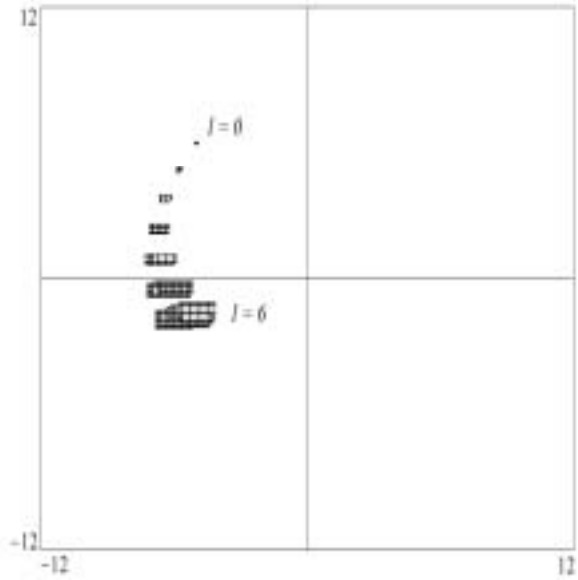


Figure 6-4: Premier exemple. Projection de l'état sur le plan (x, y).

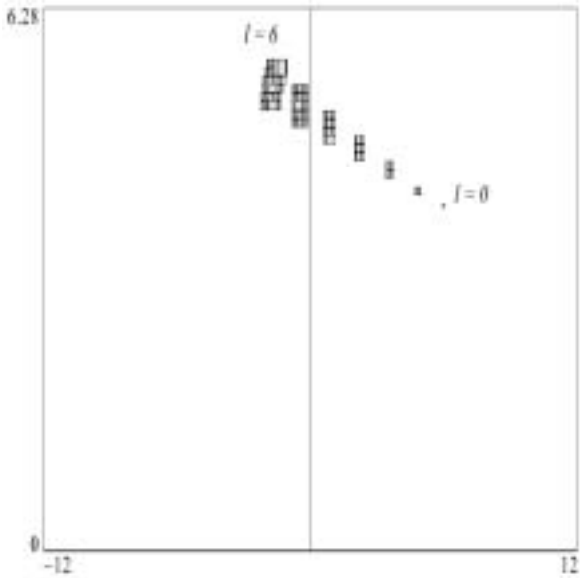


Figure 6-5: Premier exemple. Projection de l'état sur le plan (y, θ).

La taille de l'ensemble correspondant à l'état prédit croît à chaque pas, comme on pouvait s'y attendre, à cause de l'accumulation des erreurs de prédiction. Ce phénomène est aggravé par le fait que ImageSp ne donne qu'une approximation extérieure de l'ensemble des états potentiels. Ainsi, au pas $l = 6$, le sous-pavage contenant tous les états possibles est contenu dans le pavé $[-6.85\text{ m}, -4.12\text{ m}] \times [-2.25\text{ m}, -1.03\text{ m}] \times [5.10\text{ rad}, 5.70\text{ rad}]$. Le volume d'incertitude a été multiplié par 2000 en 6 pas. Ce premier exemple illustre la difficulté de réaliser un suivi sans prise en compte d'un minimum d'observations pour corriger les prédictions effectuées.

6.3.2 Second exemple : suivi avec correction

Les conditions expérimentales (carte, configuration initiale, vitesse des roues) de ce second exemple sont les mêmes que dans le premier, mais à chaque pas l, l'ensemble prédit est corrigé à l'aide de la prise en compte des observations. L'algorithme de suivi est appliqué pour $lT \in [0\text{ s}, 26\text{ s}]$. A chaque pas l, le sous-pavage prédit $\widehat{\mathcal{X}_l^+}$ est évalué grâce à ImageSp, et le sous-pavage corrigé $\mathcal{X}_{l+1}$ grâce à SiviaSp en utilisant le test masqué $t_{\text{ideal}[]}(.)$ (cf. paragraphe 4.4.2). Les sous-pavages corrigés sont représentés sur les figures 6-6 et 6-7.

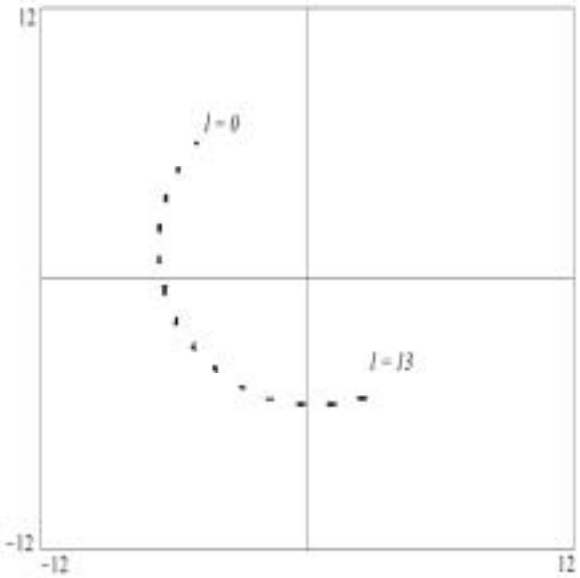


Figure 6-6: Second exemple. Projection de l'état corrigé sur le plan (x, y).

L'évaluation des 13 pas a duré 10 s, soit moins que la durée de l'expérience simulée. Dans cet exemple, la prise en compte d'observations permet aux incertitudes sur la configuration du robot de demeurer dans des bornes raisonnables. Il serait très facile d'améliorer ces bornes en réduisant le facteur de précision ϵ, mais cette amélioration de la qualité des ensembles évalués se ferait au prix d'un temps de calcul plus élevé. Sur la figure 6-7, l'état semble faire un saut, mais cela est simplement dû au fait que θ est contraint à rester dans l'intervalle $[0, 2\pi]$.

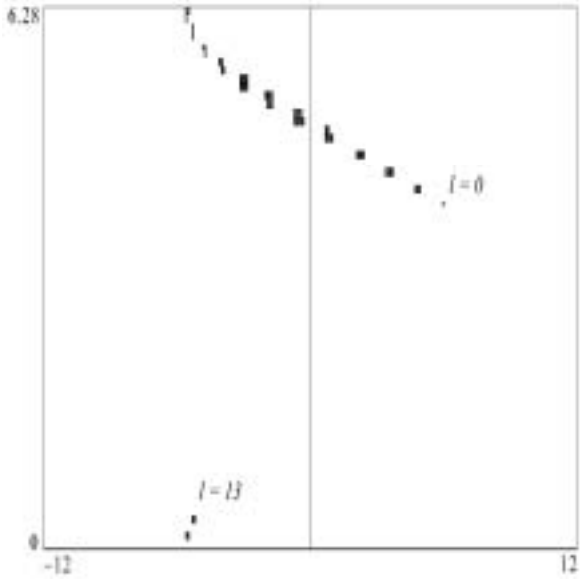


Figure 6-7: Second exemple. Projection de l'état corrigé sur le plan (y, θ).

6.3.3 Troisième exemple : suivi d'hypothèses multiples

Ce troisième exemple illustre la possibilité de réaliser un suivi de plusieurs hypothèses sur la configuration du robot sans modifier l'algorithme. La pièce dans laquelle se trouve le robot est la même que précédemment. Par contre, la configuration réelle du robot devient $(x_c, y_c, \theta)_0 = (6, 6.5, \pi)$. La figure 6-8 représente deux états possibles appartenant au sous-pavage $\widehat{\mathcal{X}_0}$ correspondant à l'ensemble des états initiaux compatibles avec les mesures faites lors de la localisation statique. Ce sous-pavage est constitué de deux sous-ensembles disjoints. Une telle situation est due à la symétrie de la pièce dans laquelle se trouve le robot.

L'algorithme de suivi est appliqué avec $v_g = R\omega_g = 0.3$ m.s^{-1} et $v_d = R\omega_d = 0.3$ m.s^{-1}, ce qui correspond à un déplacement en ligne droite. La figure 6-9 représente la projection sur le plan (x, y) des sous-pavages contenant de manière garantie la véritable configuration à chaque pas d'évaluation.

Chacun des deux sous-ensembles admissibles est suivi jusqu'à ce qu'une information soit disponible pour prouver que l'un d'entre eux ne contient en fait pas d'état compatible avec les mesures. Seuls les ensembles corrigés sont représentés. La figure 6-10 illustre les mesures disponible au pas $l = 9$, lorsque la configuration située dans le demi plan inférieur est éliminée.

L'algorithme a effectué 20 prédictions-corrections successives en 17 s. Tant que le volume des sous-pavages reste petit (ce qui signifie que la configuration est assez bien connue), les étapes de prédiction et de correction sont relativement rapides.

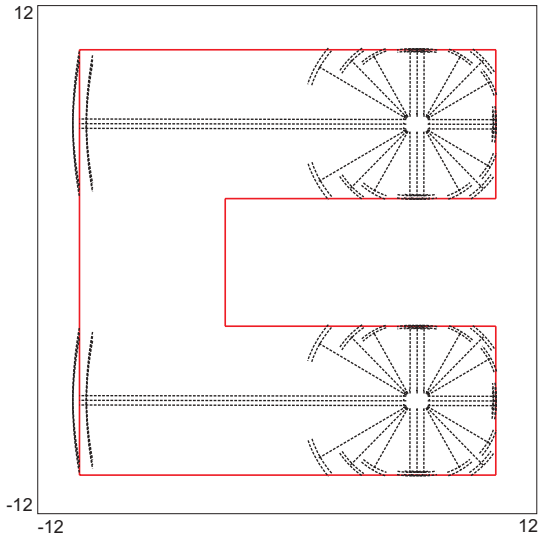


Figure 6-8: Troisième exemple. Deux configurations possibles appartenant à l'ensemble $\widehat{\mathcal{X}_0}$.

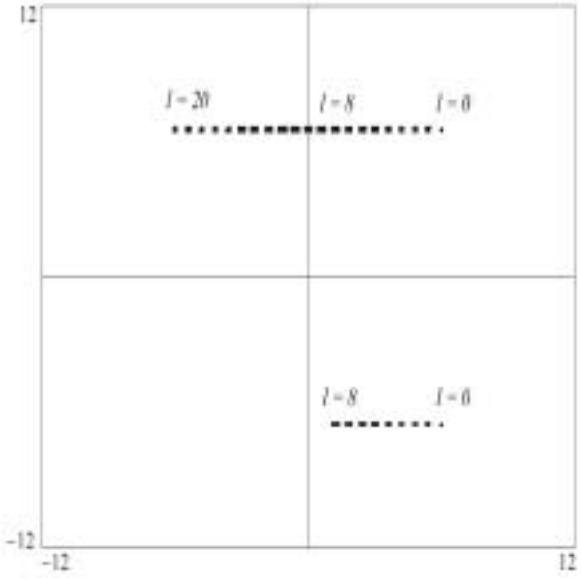


Figure 6-9: Troisième exemple. Projection sur le plan (x, y). L'un des sous-ensembles disparaît au pas $l = 9$.

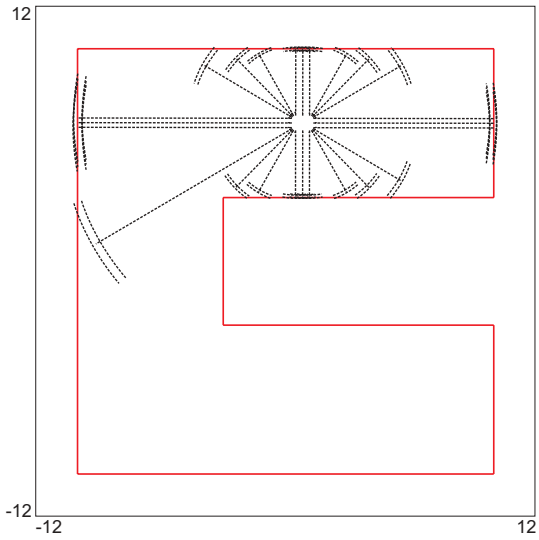


Figure 6-10: Troisième exemple. Pas 9, qui lève l'ambiguité.

6.3.4 Quatrième exemple : présence de données aberrantes

Dans cet exemple, les conditions expérimentales des deux premiers exemples sont reprises. La véritable configuration initiale est ainsi $(x_c, y_c, \theta)_0 = (-5 \text{ m}, 6 \text{ m}, 4 \text{ rad})$ et les vitesses des roues sont $v_g = R\omega_g = 0.65 \text{ m.s}^{-1}$ et $v_d = R\omega_d = 0.75 \text{ m.s}^{-1}$. Mais maintenant, des données aberrantes sont introduites. Pour cela, 8 capteurs au *maximum* sont sélectionnés, une moitié aura sa mesure annulée (panne d'un capteur) et l'autre verra sa mesure multipliée par 1.5 (réflexions multiples). La carte reste identique à celle utilisée dans les cas précédents, elle est toujours supposée exacte.

La version robustifiée de l'algorithme de localisation est utilisée pour l'étape de correction. Le test employé est $t_{\text{outliers}[]}(., q)$ (cf. paragraphe 4.4.2) qui permet de tolérer q données aberrantes. Ici, q sera fixé à 8 (un *tiers* de données aberrantes sont tolérées).

La localisation initiale est réalisée en 80 s. Le suivi est effectué jusqu'au pas $l = 14$ en 18 s. Comme un certain nombre de données aberrantes sont tolérées, deux ensembles disjoints contiennent la configuration initiale réelle au pas $l = 0$. Ces deux ensembles sont suivis jusqu'à ce que les mesures permettent d'en éliminer un au pas $l = 5$. Les figures 6-11 et 6-12 illustrent pour quelques pas les résultats obtenus par l'algorithme de suivi du robot. Sur la colonne de gauche est représentée l'évolution de la projection sur le plan (x, y) de l'ensemble des configurations ; sur la colonne de droite, le robot ainsi que les mesures effectuées sont représentés dans une (ou deux) configuration(s) appartenant au sous-pavage corrigé. Lorsque le sous-pavage corrigé est composé de deux sous-ensembles disjoints, deux configurations (une proche de la configuration réelle et une configuration *fantôme*) appartenant à chacun de ces sous-ensembles sont représentées. A partir du pas $l = 5$, il n'y a plus qu'un seul ensemble à détailler.

Cet exemple montre un comportement de l'algorithme de suivi tout à fait satisfaisant, même en présence d'un nombre important de données aberrantes. Si la localisation initiale prend un temps plus

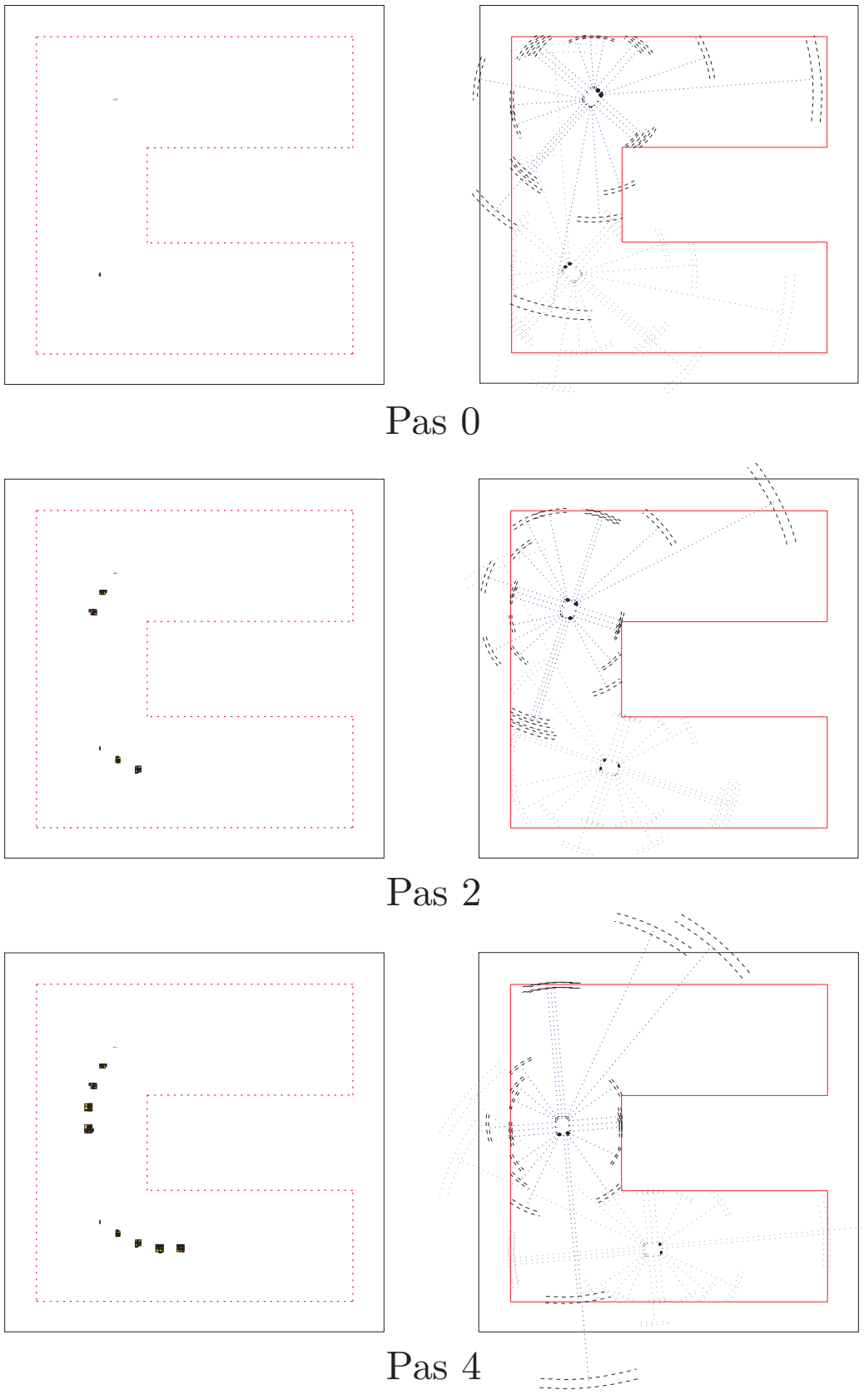
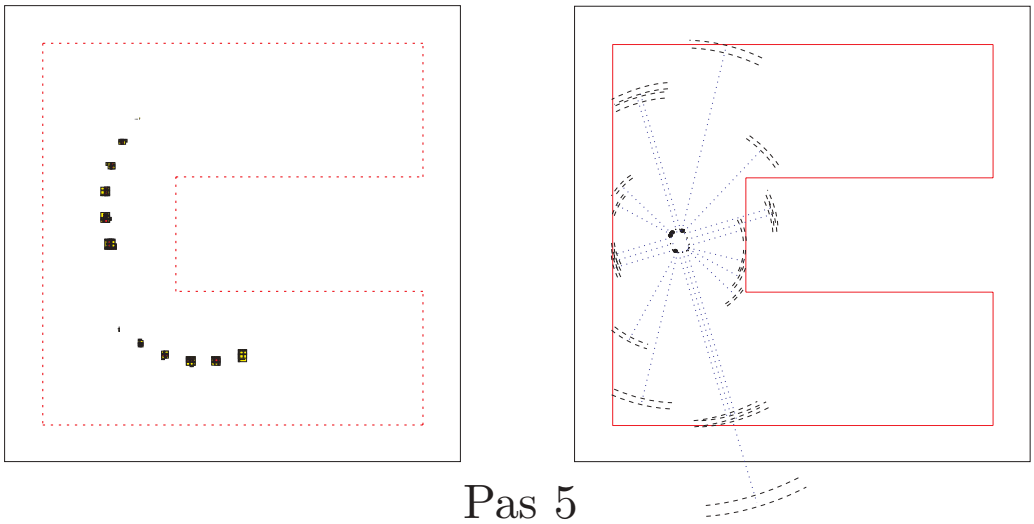
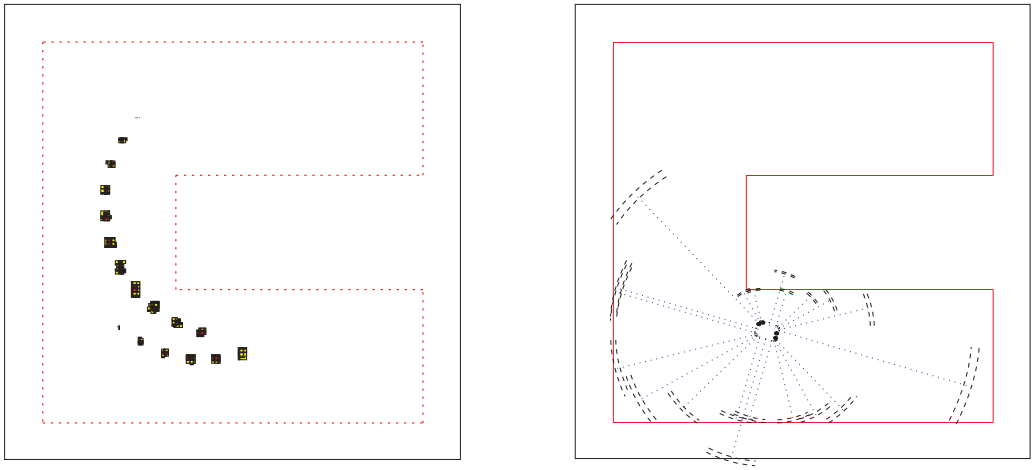


Figure 6-11: Evolution du robot (première partie). Il existe une ambiguité.





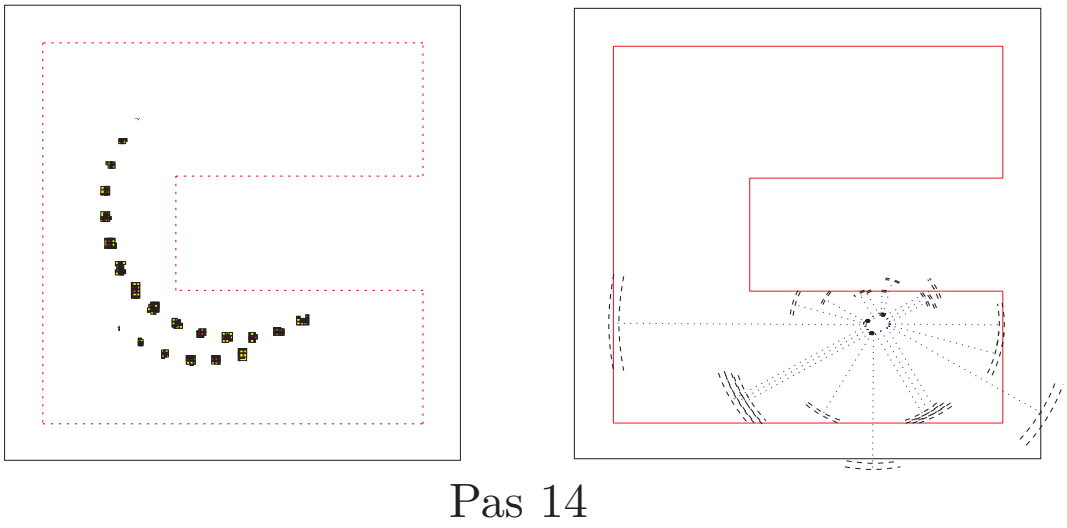


Figure 6-12: Evolution du robot (seconde partie). L'ambiguïté a disparu.

important qu'en absence de données aberrantes, le temps de calcul pour le suivi n'est pas significativement accru.

6.4 Conclusions

Les principales propriétés de l'algorithme récursif d'estimation d'état utilisant les outils de l'analyse par intervalle présenté au chapitre 5 ont été illustrées sur plusieurs exemples non-triviaux de poursuite de robot mobile. A chaque instant d'estimation, cet algorithme possède la propriété de fournir un ensemble contenant de manière garantie toutes les valeurs de l'état qui sont compatibles avec les mesures fournie par les capteurs du robot.

Cette procédure de localisation et de suivi ne souffre pas des inconvénients de beaucoup d'autres méthodes antérieures. Elle fournit une solution garantie, contrairement aux méthodes basée sur le filtrage de Kalman étendu (voir [Crowley89], [Leonard et al.91] et [Leonard et al.92]) ou les techniques à erreur bornée nécessitant une linéarisation préalable, telles que celles décrites dans [Bertsekas et al.71], [Schweppe73], [Maksarov et al.96] et [Durieu et al.96b]. Il n'est pas nécessaire de faire de pré-traitement afin d'associer mesures et environnement comme cela est fait dans [Drumheller87] et [Grimson et al.87]. L'algorithme de suivi a en outre la possibilité de suivre sans aucune modification plusieurs hypothèses de localisation du robot, contrairement à [Halbwachs et al.97]. Il ne nécessite pas de phase d'initialisation préalable comme les techniques décrites dans [Neira et al.96] et [Leonard et al.91]. Enfin, il est extrêmement robuste à l'égard de données aberrantes, qui sont à peu près inévitables lorsqu'il s'agit de réaliser une localisation en utilisant des données télémétriques obtenues avec des capteurs à ultrasons..

La version présentée est pratiquement prête pour une adaptation au suivi en temps réel d'un robot mobile. Quelques aménagements sont encore nécessaires pour rendre opérationnelle cette première version très rudimentaire d'un estimateur robuste. Tout d'abord, il faudra pouvoir tenir compte d'un nombre variable de données aberrantes, en basant l'étape de correction sur un véritable GOMNE afin de ne pas tolérer plus de données aberrantes que nécessaire (risque d'apparition de configurations fantômes).

Chapitre 7

Conclusion et perspectives

Développée initialement pour quantifier et maîtriser la précision des calculs effectués sur des ordinateurs, l'analyse par intervalles a rapidement permis d'apporter des solutions originales à des problèmes mathématiques classiques tels l'optimisation de fonctions, la résolution de systèmes d'équations linéaires et non-linéaires, etc. L'originalité de l'approche est que les résultats obtenus sont globaux et garantis.

L'intérêt récent accordé à l'analyse par intervalles en automatique a plus ou moins coïncidé avec l'apparition de langages orientés-objet puissants permettant une manipulation des intervalles avec la même facilité que celle des entiers ou des réels en virgule flottante. La plus grande partie de la littérature en analyse par intervalles appliquée à l'automatique concerne des problèmes de caractérisation de domaines de stabilité de systèmes ou de synthèse de correcteurs robustes, l'application à l'identification de paramètres de modèles non-linéaires est plus marginale, et l'estimation d'état quant à elle est restée, à notre connaissance, jusqu'à maintenant pratiquement absente. Nous nous sommes donc intéressé à ces deux aspects de l'automatique.

Dans la première partie de cette thèse, l'estimation des paramètres de modèles non-linéaires a été envisagée sous deux points de vue différents. On peut rechercher toutes les valeurs des paramètres correspondant à l'optimum d'une fonction coût mettant en jeu les grandeurs mesurées sur le dispositif réel à modéliser et leur pendant prédit par le modèle. Les valeurs des paramètres obtenues sont alors optimales au sens de la fonction coût. L'approche à erreurs bornées consiste quant à elle à caractériser l'ensemble des valeurs admissibles des paramètres, c'est-à-dire telles que les sorties prédites par le modèle restent à l'intérieur d'intervalles englobant les sorties mesurées. L'estimée n'est alors plus constituée d'un ou plusieurs points isolés mais d'un ensemble.

Le chapitre 3 a décrit une solution utilisant l'analyse par intervalles dans chacun de ces cas. Ainsi, lorsque l'on considère l'optimisation d'une fonction coût, l'algorithme d'optimisation globale de Hansen fournit de manière *garantie*, contrairement à de nombreuses méthodes classiques, un encadrement de *tous* les arguments de son optimum global. A travers un exemple, l'utilisation d'une telle technique a permis de mettre en évidence un problème d'identifiabilité, sans faire d'étude spécifique préalable sur le modèle.

Dans le contexte d'erreurs bornées, l'algorithme d'inversion ensembliste utilisant l'analyse par intervalles Sivia a déjà démontré ses possibilités. L'ensemble des paramètres admissibles est encadré par deux sous-pavages, c'est-à-dire par deux unions de pavés disjoints. Une nouvelle version récursive de cet algorithme, mettant en jeu la notion de test d'inclusion, composé d'un certain nombre de tests élémentaires, a été introduite. La prise en compte de données aberrantes devient alors très simple à mettre en œuvre. L'adéquation d'une sortie prédite à une mesure s'exprime en général par un test élémentaire, l'adéquation du modèle à toutes les mesures correspond à la satisfaction de n tests élémentaires. Lorsqu'on souhaite tenir compte de données aberrantes, au lieu de satisfaire n tests, il suffit d'en satisfaire $n - q$ où q est le nombre de données aberrantes tolérées. Les vecteurs de masques, contenant les résultats des tests élémentaires ont ensuite été introduits. Grâce à cette mise en mémoire de résultats intermédiaires, il est possible de réduire de manière très significative le nombre de tests à effectuer pour réaliser une inversion ensembliste.

Jusqu'à présent, lors de l'utilisation d'algorithmes d'inversion ensembliste par analyse par intervalles dans le contexte d'erreurs bornées, les bornes sur les erreurs étaient supposées connues. Lorsque ce n'est pas le cas, nous avons développé l'algorithme Meboe, qui évalue la plus petite borne d'erreur fournissant un ensemble de vecteurs de paramètres admissibles non vide.

Les perspectives d'amélioration des performances de la méthode d'identification par minimisation d'un critère étant essentiellement liées à l'amélioration de l'algorithme d'optimisation, elles nous semble relativement restreintes. Par contre, l'estimation dans le cadre d'erreurs bornées devrait encore permettre des développements intéressants. En particulier pour ce qui concerne Meboe, une version très simpliste a été donnée et une implantation beaucoup plus efficace, combinant recherche ensembliste et recherche ponctuelle devrait être envisagée. En outre, les propriétés théoriques de cet estimateur devraient être établies, en particulier en ce qui concerne sa robustesse.

Les sous-pavages solutions de l'algorithme Sivia dans ses premières versions présentaient l'inconvénient de n'être qu'une union de pavés sans agencement particulier, ce qui rendait l'ensemble solution difficile à manipuler. Nous avons proposé l'organisation de ces sous-pavages au sein d'arbres binaires. Cette méthode permet de bâtir des algorithmes récursifs très concis pour la manipulation de sous-pavages comme représentants d'ensembles plus complexes. L'extension de Sivia à l'inversion ensembliste d'ensembles quelconques pouvant être décrits par des sous-pavages, le calcul d'image directe d'un sous-pavage par ImageSp sont ainsi facilités. Les propriétés de convergence de ce dernier ont été établies.

Grâce à ImageSp, à Sivia et à l'usage des sous-pavages, il a été possible de bâtir un algorithme récursif d'estimation d'état garanti présenté au chapitre 5. Cet algorithme a une structure globale analogue à celui du filtre de Kalman, mais dans un contexte d'erreurs bornées ; à tout instant un ensemble contenant les valeurs de l'état compatibles avec les informations disponibles est fourni. Une phase de prédiction (réalisée à l'aide de ImageSp ou dans certaines conditions de Sivia) alterne avec une phase de correction (utilisant Sivia) qui tient compte des données observées.

Cette méthode a été appliquée à la localisation puis au suivi d'un robot, présentés respectivement aux chapitres 4 et 6. Les propriétés de l'algorithme d'estimation d'état ont été mises en évidence. Tout d'abord, compte-tenu des hypothèses de modélisation du problème, il est possible d'enfermer l'ensemble des configurations compatibles avec les mesures et les bornes admises sur les bruits de mesure dans un sous-pavage. La présence de données aberrantes, comme les ambiguïtés liées aux symétries de la pièce dans laquelle se trouve le robot ont été prises en compte sans difficulté. Nous avons vu que des ensembles de configurations possibles disjoints pouvaient être considérées et que le traitement de chacune de ces configurations ne posait aucun problème, l'algorithme de localisation ou de suivi, traitant des ensembles compacts quelconques, reste inchangé. En outre, nous avons vu que le suivi en présence de données aberrantes pouvait se faire en temps réel sur les exemples considérés.

La méthode de suivi est prête pour une mise en œuvre sur un robot réel. L'étape suivante, si l'algorithme de suivi en situation réelle présente un comportement satisfaisant, est de réaliser une planification de trajectoire pour le robot en tenant compte d'ambiguïtés sur sa localisation. Il faut privilégier un sous-ensemble, établir une commande pour faire évoluer le robot. Cette commande doit prendre garde à ne pas mettre en péril le robot s'il ne se trouve pas dans la configuration pour laquelle elle a été calculée, par exemple il faut éviter d'envoyer le robot contre un mur. Autant l'étape de vérification est assez simple à mettre en œuvre grâce à l'algorithme de calcul d'image directe `SpImage`, autant l'étape de planification est délicate, mais une littérature assez étoffée existe dans ce domaine et des solutions utilisant l'analyse par intervalles sont en cours d'étude.

Une mise à disposition des exécutables présentés dans le cadre de cette thèse pour les problèmes d'inversion ensembliste ainsi que pour le problème du suivi du robot est prévue à l'adresse suivante

`http: //supelec.supelec.fr/lss/fr/personnels/kieffer/`

Annexe A

Notations

Les vecteurs sont représentés en gras et sont surmontés d'une flèche : $\vec{\mathbf{u}}$. Les points sont en gras : $\mathbf{a}, \mathbf{b}, \mathbf{c}$. Les coordonnées des vecteurs $\vec{\mathbf{u}}$ et des points $\mathbf{a}$ du plan sont notées x_u, y_u et x_a, y_a. Le tableau A.1 rappelle une partie des notations introduites dans ce document, en particulier celles introduites dans le problème de robotique.

α_i	:	incertitude relative de mesure du capteur i
$(\mathbf{a}, \mathbf{b})$	:	droite passant par $\mathbf{a}$ et $\mathbf{b}$,
$[\mathbf{a}, \mathbf{b}]$	:	segment ayant pour extrémités $\mathbf{a}$ et $\mathbf{b}$,
$\overrightarrow{\mathbf{ab}}$	:	vecteur ayant pour extrémités $\mathbf{a}$ et $\mathbf{b}$,
$\tilde{\gamma}$	:	demi-ouverture du cône d'émission,
d	:	voie (distance entre les roues motrices),
$d_i, [d_i]$	:	mesure et intervalle mesure fournis par le capteur i,
i	:	index pour les capteurs,
IR	:	ensemble des intervalles réels,
IB	:	ensemble des intervalles booléens,
j	:	index pour les segments,
$d(\mathbf{s}, (\mathbf{a}, \mathbf{b}))$	:	distance orthogonale entre $\mathbf{s}$ et $(\mathbf{a}, \mathbf{b})$,
$d_{\vec{\mathbf{u}}}(\mathbf{s}, (\mathbf{ab}))$	:	distance entre $\mathbf{s}$ et $(\mathbf{a}, \mathbf{b})$ suivant $\vec{\mathbf{u}}$,
n_{s}	:	nombre de capteurs,
n_{w}	:	nombre de segments,
$\mathbf{p} = (x_c, y_c, \theta)^{\mathrm{T}}$	:	configuration du robot,
$\mathcal{P}_{\mathrm{int}}, \mathcal{P}_{\mathrm{ext}}$	:	partition de l'espace en deux sous-ensembles,
$\mathcal{R}$	:	repère du robot,
$\mathcal{S}$	:	ensemble des configurations cohérentes,
$(\mathbf{s}, \vec{\mathbf{u}})$	:	droite passant par $\mathbf{s}$ et de vecteur directeur $\vec{\mathbf{u}}$,
$\langle \vec{\mathbf{u}}, \vec{\mathbf{v}} \rangle$	:	produit scalaire de $\vec{\mathbf{u}}$ et $\vec{\mathbf{v}}$,
$\mathcal{W}$	:	repère du laboratoire,
$\wedge$	:	ET logique,
$\vee$	:	OU logique.

Tableau A.1: Notations.

Annexe B

Espacement

Dans cette annexe est présentée la fonction d'évaluation de l'espacement en utilisant les notions introduites au chapitre 4. Dans une seconde partie, on peut trouver la fonction d'inclusion de la fonction initiale. Comme la fonction de calcul de l'espacement utilise des branchements conditionnels, la fonction d'inclusion utilise la fonction χ pour tenir compte de ces branchements (cf. chapitre 2).

B.1 Evaluation de l'espacement

La configuration du robot est $\mathsf{p} = (x_c, y_c, \theta)^{\mathrm{T}}$. Considérons un capteur $\tilde{\mathsf{s}} = (\tilde{x}, \tilde{y})^{\mathrm{T}}$ ainsi que son cône d'émission $\mathcal{C} = \mathcal{C}(\mathsf{s}, \overrightarrow{\mathsf{u_1}}, \overrightarrow{\mathsf{u_2}})$. Dans le repère du laboratoire

$$\mathsf{s} = \begin{pmatrix} x_c \\ y_c \end{pmatrix} + \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix}, \tag{B.1}$$

$$\overrightarrow{\mathsf{u_1}} = \begin{pmatrix} \cos\left(\theta + \tilde{\theta} - \tilde{\gamma}\right) \\ \sin\left(\theta + \tilde{\theta} - \tilde{\gamma}\right) \end{pmatrix}, \quad \overrightarrow{\mathsf{u_2}} = \begin{pmatrix} \cos\left(\theta + \tilde{\theta} + \tilde{\gamma}\right) \\ \sin\left(\theta + \tilde{\theta} + \tilde{\gamma}\right) \end{pmatrix} \tag{B.2}$$

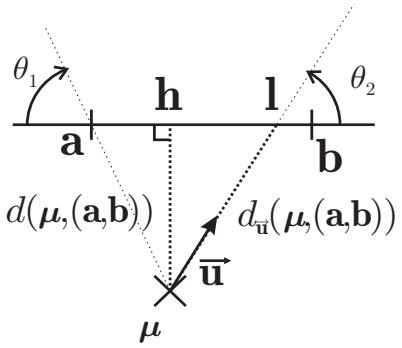


Figure B-1: Distances d'un point à une droite.

La distance entre un point μ et la droite (a, b) suivant le vecteur unitaire $\vec{u}$ est

$$\begin{aligned} d_{\vec{u}}(\mu,(a,b)) &= \left\|\overrightarrow{af}\right\| = \frac{\left\|\overrightarrow{ah}\right\|}{|\sin\theta_2|} = \frac{\|\overrightarrow{a\mu}\|\,|\sin\theta_1|}{|\sin\theta_2|} = \frac{\left\|\overrightarrow{ab}\right\| . \|\overrightarrow{a\mu}\| . |\sin\theta_1|}{\left\|\overrightarrow{ab}\right\| . \|\vec{u}\| . |\sin\theta_2|} \\ d_{\vec{u}}(\mu,(a,b)) &= \frac{\left|\det\left(\overrightarrow{ab},\overrightarrow{a\mu}\right)\right|}{\left|\det\left(\overrightarrow{ab},\vec{u}\right)\right|}, \end{aligned} \tag{B.3}$$

et la distance entre μ et la droite (a, b) est

$$\begin{aligned} d(\mu,(a,b)) &= \left\|\overrightarrow{ah}\right\| = \|\overrightarrow{a\mu}\| . |\sin\theta_1| = \frac{\left\|\overrightarrow{ab}\right\| . \|\overrightarrow{a\mu}\| . |\sin\theta_1|}{\left\|\overrightarrow{ab}\right\|} \\ d(\mu,(a,b)) &= \frac{\left|\det\left(\overrightarrow{ab},\overrightarrow{a\mu}\right)\right|}{\left\|\overrightarrow{ab}\right\|}. \end{aligned} \tag{B.4}$$

Le tableau B.1 présente l'algorithme de calcul de l'espacement entre un segment $[a, b]$, pris isolément et un cône $\mathcal{C}(s, \vec{u_1}, \vec{u_2})$.

$r(s, \vec{u_1}, \vec{u_2}, a, b)$

if $\left(\det\left(\overrightarrow{ab},\overrightarrow{as}\right) \geq 0\right)$
    return $(+\infty)$;

if $\left(\left\langle\overrightarrow{ab},\overrightarrow{sa}\right\rangle \leq 0\right) \wedge \left(\left\langle\overrightarrow{ab},\overrightarrow{sb}\right\rangle \geq 0\right) \wedge \left(\left\langle\overrightarrow{ab},\vec{u_1}\right\rangle \leq 0\right) \wedge \left(\left\langle\overrightarrow{ab},\vec{u_2}\right\rangle \geq 0\right)$
    $r_h = d(s,(a,b))$ else $r_h = +\infty$;

if$(\det(\vec{u_1},\overrightarrow{sa}) \geq 0) \wedge (\det(\vec{u_2},\overrightarrow{sa}) \leq 0)$
    $r_a = \|\overrightarrow{sa}\|$ else $r_a = +\infty$;

if $\left(\det\left(\vec{u_1},\overrightarrow{sb}\right) \geq 0\right) \wedge \left(\det\left(\vec{u_2},\overrightarrow{sb}\right) \leq 0\right)$
    $r_b = \left\|\overrightarrow{sb}\right\|$ else $r_b = +\infty$;

for $i = 1$ to 2
    if$(\det(\overrightarrow{sa},\vec{u_i}) \geq 0) \wedge \left(\det\left(\overrightarrow{sb},\vec{u_i}\right) \leq 0\right)$
        $r_{h_i} = d_{\vec{u_i}}(s,(a,b))$ else $r_{h_i} = +\infty$;

return$(\min(r_h, r_a, r_b, r_{h_1}, r_{h_2}))$;

Tableau B.1: Evaluation de l'espacement.

B.2 Fonction d'inclusion de l'espacement

On considère un pavé contenant la configuration du robot $[\mathsf{p}] = ([x_c], [y_c], [\theta])^{\mathrm{T}}$. Le pavé $[\mathsf{s}]$ contenant la position d'un capteur s donné est évalué en remplaçant les occurrences ponctuelles par les occurrences intervalles des variables. Il en est de même pour les caractéristiques du cône $[\mathcal{C}] = \mathcal{C}\left([\mathsf{s}], \overrightarrow{[\mathsf{u_1}]}, \overrightarrow{[\mathsf{u_2}]}\right)$, le calcul des distances entre un pavé et une droite, et entre un pavé et une droite suivant un vecteur unitaire intervalle.

Le tableau B.2 présente la fonction d'inclusion permettant de calculer l'espacement entre un segment $[\mathsf{a}, \mathsf{b}]$, pris isolément, et un cône intervalle $[\mathcal{C}] = \mathcal{C}\left([\mathsf{s}], \overrightarrow{[\mathsf{u_1}]}, \overrightarrow{[\mathsf{u_2}]}\right)$.

$$\boxed{r\left([\mathsf{s}], \overrightarrow{[\mathsf{u_1}]}, \overrightarrow{[\mathsf{u_2}]}, \mathsf{a}, \mathsf{b}\right)}$$

$[t_1] = \det\left(\overrightarrow{\mathsf{ab}}, \overrightarrow{\mathsf{a}\,[\mathsf{s}]}\right)$;
if $\left(\underline{t_1} \geq 0\right)$
  return $(+\infty)$;

$[t_\mathsf{h}] = \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{[\mathsf{s}]\,\mathsf{a}} \right\rangle \leq 0\right) \wedge \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{[\mathsf{s}]\,\mathsf{b}} \right\rangle \geq 0\right) \wedge \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{[\mathsf{u_1}]} \right\rangle \leq 0\right) \wedge \left(\left\langle \overrightarrow{\mathsf{ab}}, \overrightarrow{[\mathsf{u_2}]} \right\rangle \geq 0\right)$;
$[r_\mathsf{h}] = \chi\left([t_\mathsf{h}], d_{[]}\left([\mathsf{s}], (\mathsf{a}, \mathsf{b})\right), +\infty\right)$;

$[t_\mathsf{a}] = \left(\det\left(\overrightarrow{[\mathsf{u_1}]}, \overrightarrow{[\mathsf{s}]\,\mathsf{a}}\right) \geq 0\right) \wedge \left(\det\left(\overrightarrow{[\mathsf{u_2}]}, \overrightarrow{[\mathsf{s}]\,\mathsf{a}}\right) \leq 0\right)$;
$[r_\mathsf{a}] = \chi\left([t_\mathsf{a}], \left\|\overrightarrow{[\mathsf{s}]\,\mathsf{a}}\right\|, +\infty\right)$;

$[t_\mathsf{b}] = \left(\det\left(\overrightarrow{[\mathsf{u_1}]}, \overrightarrow{[\mathsf{s}]\,\mathsf{b}}\right) \geq 0\right) \wedge \left(\det\left(\overrightarrow{[\mathsf{u_2}]}, \overrightarrow{[\mathsf{s}]\,\mathsf{b}}\right) \leq 0\right)$;
$[r_\mathsf{b}] = \chi\left([t_\mathsf{b}], \left\|\overrightarrow{[\mathsf{s}]\,\mathsf{b}}\right\|, +\infty\right)$;

for $i = 1$ to 2
  $[t_{\mathsf{h}_\mathsf{i}}] = \left(\det\left(\overrightarrow{[\mathsf{s}]\,\mathsf{a}}, \overrightarrow{[\mathsf{u}_i]}\right) \geq 0\right) \wedge \left(\det\left(\overrightarrow{[\mathsf{s}]\,\mathsf{b}}, \overrightarrow{[\mathsf{u}_i]}\right) \leq 0\right)$;
  $[r_{\mathsf{h}_\mathsf{i}}] = \chi\left([t_{\mathsf{h}_\mathsf{i}}], d_{\overrightarrow{[\mathsf{u_i}]}}\left([\mathsf{s}], (\mathsf{a}, \mathsf{b})\right), +\infty\right)$;

return $\left(\min\left([r_\mathsf{h}], [r_\mathsf{a}], [r_\mathsf{b}], [r_{\mathsf{h}_1}], [r_{\mathsf{h}_2}]\right)\right)$;

Tableau B.2: Fonction d'inclusion de l'espacement.

Remarque B.2.1 *Le minimum de deux intervalles est défini de la manière suivante*

$$\min\left([a], [b]\right) = \left[\min\left(\underline{a}, \underline{b}\right), \min\left(\overline{a}, \overline{b}\right)\right].$$

L'extension à un plus grand nombre d'arguments est immédiate. ♢

Annexe C

Justification de in_room

Cette annexe présente une démonstration de la proposition 3 introduite p. 91.

Proposition 7 (rappel) *Considérons un point* $\mu \in \mathbf{R}^2$ *et une carte* $\mathcal{M}$ *définissant un contour fermé* $\mathcal{M}'$ *tel que* $\mu \notin \mathcal{M}'$. μ *est à l'intérieur de* $\mathcal{M}'$ *si et seulement si* $\sum_{k=1}^{p}\left(\sum_{l=1}^{n_p}\arg\left(\overrightarrow{\mu a_{kl}},\overrightarrow{\mu b_{kl}}\right)\right)=2\pi.\diamondsuit$

Preuve : on se place dans le plan complexe et on note $z_\mu, z_{a_{11}}, \ldots$ l'affixe des points $\mu, a_{11}, \ldots$ D'après le théorème des résidus ,

$$I(z_\mu)=\mathrm{Im}\left(\int_{\mathcal{M}'}\frac{1}{z-z_\mu}dz\right)=2\pi \text{ si et seulement si } \mu \text{ est l'intérieur de } \mathcal{M}'.$$

Montrons tout d'abord que $I(z_\mu)$ est la somme des intégrales sur chacun des polygones de la carte.

$$\begin{aligned}
I(z_\mu) &= \mathrm{Im}\left(\int_{\mathcal{M}'}\frac{1}{z-z_\mu}dz\right)\\
&= \mathrm{Im}\left(\int_{\mathcal{M}'_1}\frac{1}{z-z_\mu}dz\right.\\
&\quad +\int_{b_{1n_1}a_{21}}\frac{1}{z-z_\mu}dz+\int_{\mathcal{M}'_2}\frac{1}{z-z_\mu}dz+\int_{b_{2n_2}b_{1n_1}}\frac{1}{z-z_\mu}dz\\
&\quad +\ldots\\
&\quad \left.+\int_{b_{1n_1}a_{p1}}\frac{1}{z-z_\mu}dz+\int_{\mathcal{M}'_p}\frac{1}{z-z_\mu}dz+\int_{b_{pn_p}b_{1n_1}}\frac{1}{z-z_\mu}dz\right).
\end{aligned}$$

or par hypothèse sur la carte, pour $k=1,\ldots,p$, $b_{kn_k}=a_{k1}$. Par conséquent,

$$I(z_\mu)=\sum_{k=1}^{p}I_k(z_\mu)=\sum_{k=1}^{p}\mathrm{Im}\left(\int_{\mathcal{M}'_k}\frac{1}{z-z_\mu}dz\right).$$

Il s'agit maintenant de déterminer la valeur de $I_k(z_\mu)$.

$$I_k(z_\mu) = \mathrm{Im}\left(\int_{\mathcal{M}'_k} \frac{1}{z - z_\mu} dz\right) = \sum_{k=1}^{n_k} \mathrm{Im}\left(\int_{[a_{kl}, b_{kl}]} \frac{1}{z - z_\mu} dz\right)$$

Sans nuire à la généralité du problème, on peut parfaitement supposer que μ est l'origine du repère $\mathcal{W}$ (si ce n'est pas le cas, il suffit de translater tous les vecteurs de la carte). Ainsi,

$$\int_{[a_{kl}, b_{kl}]} \frac{1}{z - z_\mu} dz = \arg_\varphi\left(\overrightarrow{\mu a_{kl}}, \overrightarrow{\mu b_{kl}}\right)$$

où $\arg_\varphi$ est la fonction argument localement holomorphe dans un voisinage de $[a_{kl}, b_{kl}]$, c'est-à-dire que la coupure passant par μ ne coupe pas le segment $[a_{kl}, b_{kl}]$ (voir figure C-1).

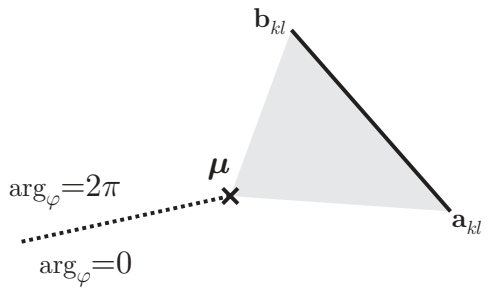


Figure C-1: Placement de la coupure (en pointillés) afin que arg soit holomorphe dans un voisinnage de $[a_{kl}, b_{kl}]$.

En outre, comme $[a_{kl}, b_{kl}]$ est un segment, (μ, a_{kl}, b_{kl}) est un triangle non dégénéré et $\arg_\varphi\left(\overrightarrow{\mu a_{kl}}, \overrightarrow{\mu b_{kl}}\right) \in]-\pi, \pi[$. On a alors

$$I_k(z_\mu) = \sum_{l=1}^{n_k} \arg\left(\overrightarrow{\mu a_{kl}}, \overrightarrow{\mu b_{kl}}\right),$$

et par conséquent

$$I(z_\mu) = \sum_{k=1}^{p} \left(\sum_{l=1}^{n_k} \arg\left(\overrightarrow{\mu a_{kl}}, \overrightarrow{\mu b_{kl}}\right)\right),$$

et ainsi

$$\sum_{k=1}^{p} \left(\sum_{l=1}^{n_k} \arg\left(\overrightarrow{\mu a_{kl}}, \overrightarrow{\mu b_{kl}}\right)\right) = 2\pi \text{ si et seulement si } \mu \text{ est l'intérieur de } \mathcal{M}'. \blacksquare$$

Annexe D

Convergence de ImageSp

La démonstration de la proposition 6, p. 121 passe par celle du lemme suivant.

Lemme D.0.1 *Si* $\mathsf{f}_{[]}$ *est une fonction d'inclusion de* $\mathsf{f}:\mathbf{R}^n \to \mathbf{R}^p$ *qui est lipschitzienne sur* $\widehat{\mathcal{X}} \subset \mathbf{R}^n$*, alors pour tout* $\alpha > 0$*, il existe* ϵ *tel que pour tout* $[\mathsf{x}] \subset \widehat{\mathcal{X}}$ *vérifiant* $w([\mathsf{x}]) \leq \epsilon$*,* $d\left(\mathsf{f}_{[]}([\mathsf{x}]), \mathsf{f}([\mathsf{x}])\right) \leq \alpha$*.*

Preuve : sachant que $\mathsf{f}_{[]}$ est lipschitz sur$\widehat{\mathcal{X}}$, il existe L tel que, pour tout $[\mathsf{x}] \subset \widehat{\mathcal{X}}$, $w\left(\mathsf{f}_{[]}([\mathsf{x}])\right) \leq L.w([\mathsf{x}])$. Prenons $\epsilon = \frac{\alpha}{L}$, et considérons $[\mathsf{x}]_0 \subset \widehat{\mathcal{X}}$ vérifant $w([\mathsf{x}]_0) \leq \epsilon$; nous avons alors $w\left(\mathsf{f}_{[]}([\mathsf{x}]_0)\right) \leq L.w([\mathsf{x}]_0) \leq \alpha$. Comme $\mathsf{f}([\mathsf{x}]_0) \subset \mathsf{f}_{[]}([\mathsf{x}]_0)$, le lemme 2.6.2 entraine que $d\left(\mathsf{f}([\mathsf{x}]_0), \mathsf{f}_{[]}([\mathsf{x}]_0)\right) \leq \alpha$. ■

On peut distinguer deux étapes dans l'algorithme ImageSp. Dans la première, $\widehat{\mathcal{X}}$ est divisé en p_0 pavés $[\mathsf{x}]_i$ dont la longueur maximale est inférieure à ϵ, et leur image par $\mathsf{f}_{[]}$ est stockée dans une liste de pavés images $\mathcal{L}_\epsilon^0 = \left\{\mathsf{f}_{[]}([\mathsf{x}]_i)\right\}_{i=1}^{p_0}$. Démontrons que pour un $\eta > 0$ donné, il existe ϵ tel que $d\left(\mathsf{f}\left(\widehat{\mathcal{X}}\right), \bigcup_{i=1}^{p_0} \mathsf{f}_{[]}([\mathsf{x}]_i)\right) \leq \eta/2$.

Preuve de la proposition 6 (première partie) : cette partie de la démonstration est inspirée de [Moore79] et [Neumaier90]. Soit $\eta > 0$; comme $\mathsf{f}_{[]}$ est une fonction d'inclusion de f, lipschitz sur $\widehat{\mathcal{X}}$, d'après le lemme D.0.1, il existe ϵ_1 tel que pour tout $[\mathsf{x}] \subset \widehat{\mathcal{X}}$ vérifiant $w([\mathsf{x}]) \leq \epsilon_1$, $d_0\left(\mathsf{f}_{[]}([\mathsf{x}]), \mathsf{f}([\mathsf{x}])\right) \leq \eta/2$. Comme $\widehat{\mathcal{X}} = \bigcup_{i=1}^{p_0} [\mathsf{x}]_i$, l'union des p_0 pavés de longueur inférieure à ϵ_1 et issus de $\widehat{\mathcal{X}}$, $\mathsf{f}\left(\widehat{\mathcal{X}}\right) = \bigcup_{i=1}^{p_0} \mathsf{f}([\mathsf{x}]_i) \subset \bigcup_{i=1}^{p_0} \mathsf{f}_{[]}([\mathsf{x}]_i)$. De ce fait, d'après le lemme 2.6.1,

$$d_0\left(\bigcup_{i=1}^{p_0} \mathsf{f}([\mathsf{x}]_i), \bigcup_{i=1}^{p_0} \mathsf{f}_{[]}([\mathsf{x}]_i)\right) = 0. \tag{D.1}$$

Le lemme 2.6.1 montre que $d_0\left(\mathsf{f}_{[]}([\mathsf{x}]_i), \bigcup_{j=1}^{p_0} \mathsf{f}\left([\mathsf{x}]_j\right)\right) \leq d_0\left(\mathsf{f}_{[]}([\mathsf{x}]_i), \mathsf{f}([\mathsf{x}]_i)\right) \leq \eta/2$ pour $i = 1, ..., p_0$. On a alors,

$$d_0\left(\bigcup_{i=1}^{p_0} \mathsf{f}_{[]}([\mathsf{x}]_i), \bigcup_{i=1}^{p_0} \mathsf{f}([\mathsf{x}]_i)\right) = \max_{i=1,...,p_0} d_0\left(\mathsf{f}_{[]}([\mathsf{x}]_i), \bigcup_{i=1}^{p_0} \mathsf{f}([\mathsf{x}]_i)\right) \leq \eta/2. \tag{D.2}$$

Ainsi, d'après $(D.1)$ et $(D.2)$, comme $\mathsf{f}\left(\widehat{\mathcal{X}}\right) = \bigcup_{i=1}^{p_0} \mathsf{f}\left([\mathsf{x}]_i\right)$, il vient

$$d\left(\mathsf{f}\left(\widehat{\mathcal{X}}\right), \bigcup_{i=1}^{p_0} \mathsf{f}_{[]}\left([\mathsf{x}]_i\right)\right) \leq \eta/2. \blacksquare$$

La seconde étape d'ImageSp correspond à la construction d'un sous-pavage contenant les pavés de $\mathcal{L}^0_{\epsilon_1}$ en utilisant la procédure $\mathsf{BuildSp}\left(\mathcal{L}^0_{\epsilon_1},\ [\mathsf{s}]^0,\ \epsilon_2\right)$. Nous supposons que ce sous-pavage est inclus à l'intérieur d'un pavé $[\mathsf{s}]^0$. Si le sous-pavage décrit par $\mathcal{L}^0_{\epsilon_1}$ n'est pas égal à $[\mathsf{s}]^0$, $[\mathsf{s}]^0$ est bisecté en $L\,[\mathsf{s}]^0$ et $R\,[\mathsf{s}]^0$. On applique alors récursivement $\mathsf{BuildSp}$ à $L\mathcal{L}^0_{\epsilon_1}$ et $R\mathcal{L}^0_{\epsilon_1}$, les intersections respectives de $\mathcal{L}^0_{\epsilon_1}$ avec $L\,[\mathsf{s}]^0$ et $R\,[\mathsf{s}]^0$. Le sous-pavage résultat sera l'union des deux sous-pavages fournis par $\mathsf{BuildSp}\left(L\mathcal{L}^0_{\epsilon_1},\ L\,[\mathsf{s}]^0,\ \epsilon_2\right)$ et $\mathsf{BuildSp}\left(R\mathcal{L}^0_{\epsilon_1},\ R\,[\mathsf{s}]^0,\ \epsilon_2\right)$. Nous allons montrer maintenant que pour un η donné, il existe ϵ_2 tel que $d\left(\mathsf{BuildSp}\left(\mathcal{L}^0_{\epsilon_1},\ [\mathsf{s}]^0,\ \epsilon_2\right),\ \bigcup_{i=1}^{p_0} \mathsf{f}_{[]}\left([\mathsf{x}]_i\right)\right) \leq \eta/2$.

Preuve de la proposition 6 (seconde partie) : cette seconde partie, plus spécifique, est démontrée par récurrence. La propriété sera tout d'abord établie pour des appels non récursifs à $\mathsf{BuildSp}$, c'est-à-dire lorsque $\mathsf{BuildSp}$ renvoit une feuille du sous-pavage. Nous verrons ensuite que la propriété vraie pour des feuilles le reste également pour les nœuds, jusqu'à la racine de l'arbre binaire considéré.

Pour simplifier la présentation, les listes $\mathcal{L}^k_{\epsilon_1} = \left\{[\mathsf{z}]^k_i\right\}_{i=1}^{p_0}$ seront supposées avoir toutes le même nombre de pavés p_0, dont certains peuvent être vides. Une liste sera vide si elle ne contient que des pavés vides. Dans la mise en œuvre de l'algorithme, ces pavés vides sont évidemment supprimés.

Soit $\epsilon_2 = \eta/2$. Considérons tout d'abord le cas où le $k$ième appel à $\mathsf{BuildSp}$ renvoie une feuille (traitement non récursif). Les arguments de $\mathsf{BuildSp}$ sont $\mathcal{L}^k_{\epsilon_1} = \left\{[\mathsf{z}]^k_i\right\}_{i=1}^{p_0}$, $[\mathsf{s}]^k$ et ϵ_2. Trois cas, correspondant aux trois branchements conditionnels de l'algorithme doivent être considérés :

(i) Si $\mathcal{L}^k_{\epsilon_1}$ est vide ($\bigcup_{i=1}^{p_0} [\mathsf{z}]^k_i = \emptyset$) alors $\mathsf{BuildSp}\left(\mathcal{L}^k_{\epsilon_1},\ [\mathsf{s}]^k,\ \epsilon_2\right)$ renvoit un sous-pavage vide et, par convention

$$d\left(\mathsf{BuildSp}\left(\mathcal{L}^k_{\epsilon_1},\ [\mathsf{s}]^k,\ \epsilon_2\right),\ \bigcup_{i=1}^{p_0} [\mathsf{z}]^k_i\right) = d\left(\emptyset, \emptyset\right) = 0.$$

(ii) S'il existe i tel que $[\mathsf{z}]^k_i = [\mathsf{s}]^k$, alors $\bigcup_{i=1}^{p_0} [\mathsf{z}]^k_i = [\mathsf{s}]^k$, et $\mathsf{BuildSp}\left(\mathcal{L}^k_{\epsilon_1}, [\mathsf{s}]^k, \epsilon_2\right)$ renvoient un sous-pavage correspondant à $[\mathsf{s}]^k$. De ce fait

$$d\left(\mathsf{BuildSp}\left(\mathcal{L}^k_{\epsilon_1},\ [\mathsf{s}]^k,\ \epsilon_2\right), \bigcup_{i=1}^{p_0} [\mathsf{z}]^k_i\right) = 0.$$

(iii) Enfin, si $\mathcal{L}^k_{\epsilon_1}$ ne représente pas un ensemble vide, mais si $w\left([\mathsf{s}]^k\right) \leq \epsilon_2 = \eta/2$, alors $\mathsf{BuildSp}\left(\mathcal{L}^k_{\epsilon_1},\ [\mathsf{s}]^k,\ \epsilon_2\right)$ renvoit le sous-pavage correspondant à $[\mathsf{s}]^k$.

Comme $\bigcup_{i=1}^{p_0} [\mathsf{z}]^k_i \subset \mathsf{BuildSp}\left(\mathcal{L}^k_{\epsilon_1},\ [\mathsf{s}]^k,\ \epsilon_2\right) = [\mathsf{s}]^k$, et comme $w\left([\mathsf{s}]^k\right) \leq \eta/2$, le lemme 2.6.2

implique

$$d\left(\mathsf{BuildSp}\left(\mathcal{L}^k_{\epsilon_1},\ [\mathsf{s}]^k,\ \epsilon_2\right), \bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i\right) \leq \eta/2.$$

Dans les trois cas, $d\left(\mathsf{BuildSp}\left(\mathcal{L}^k_{\epsilon_1},\ [\mathsf{s}]^k,\ \epsilon_2\right), \bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i\right) \leq \eta/2$.

Supposons maintenant qu'au $k$ième appel, $\mathsf{BuildSp}\left(\mathcal{L}^k_{\epsilon_1},\ [\mathsf{s}]^k,\ \epsilon_2\right)$ effectue un appel récursif d'elle-même et que les deux sous-pavages renvoyés par ces appels satisfont

$$d\left(\mathsf{BuildSp}\left(L\mathcal{L}^k_{\epsilon_1},\ L[\mathsf{s}]^k,\ \epsilon_2\right),\ \bigcup_{i=1}^{p_0}[\mathsf{z}]^{k,L}_i\right) \leq \eta/2,$$

et

$$d\left(\mathsf{BuildSp}\left(R\mathcal{L}^k_{\epsilon_1},\ R[\mathsf{s}]^k,\ \epsilon_2\right),\ \bigcup_{i=1}^{p_0}[\mathsf{z}]^{k,R}_i\right) \leq \eta/2,$$

avec $L[\mathsf{s}]^k \cup R[\mathsf{s}]^k = [\mathsf{s}]^k$, $\mathcal{L}^k_{\epsilon_1} = \left\{[\mathsf{z}]^k_i\right\}_{i=1}^{p_0}$, $L\mathcal{L}^k_{\epsilon_1} = \left\{[\mathsf{z}]^{k,L}_i\right\}_{i=1}^{p_0} = \left\{[\mathsf{z}]^k_i \cap L[\mathsf{s}]^k\right\}_{i=1}^{p_0}$ et $R\mathcal{L}^k_{\epsilon_1} = \left\{[\mathsf{z}]^{k,R}_i\right\}_{i=1}^{p_0} = \left\{[\mathsf{z}]^k_i \cap R[\mathsf{s}]^k\right\}_{i=1}^{p_0}$. Comme précédemment, certains $[\mathsf{z}]^{k,L}_i$ et $[\mathsf{z}]^{k,R}_i$ peuvent être vides. Il s'agit de montrer que $d\left(\mathsf{BuildSp}\left(\mathcal{L}^k_{\epsilon_1},\ [\mathsf{s}]^k,\ \epsilon_2\right),\ \bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i\right) \leq \eta/2$.
Soit $\widehat{\mathcal{S}^k_{\epsilon_2}} = \mathsf{BuildSp}\left(\mathcal{L}^k_{\epsilon_1},\ [\mathsf{s}]^k,\ \epsilon_2\right)$, $\mathcal{S}^{k,L}_{\epsilon_2} = \mathsf{BuildSp}\left(L\mathcal{L}^k_{\epsilon_1},\ L[\mathsf{s}]^k,\ \epsilon_2\right)$, et
$\mathcal{S}^{k,R}_{\epsilon_2} = \mathsf{BuildSp}\left(R\mathcal{L}^k_{\epsilon_1},\ R[\mathsf{s}]^k,\ \epsilon_2\right)$. BuildSp à l'appel k renvoit $\widehat{\mathcal{S}^{k,L}_{\epsilon_2}} \cup \mathcal{S}^{k,R}_{\epsilon_2}$, c'est-à-dire $\widehat{\mathcal{S}^k_{\epsilon_2}}$ (voir l'algorithme p. 135). Comme $[\mathsf{z}]^{k,L}_i = [\mathsf{z}]^k_i \cap L[\mathsf{s}]^k$, $[\mathsf{z}]^{k,L}_i \subset [\mathsf{z}]^k_i$ pour $i = 1, ..., p_0$, et de ce fait $\bigcup_{i=1}^{p_0}[\mathsf{z}]^{k,L}_i \subset \bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i$. Ainsi, d'après la propriété 2.6.1, $d_0\left(\widehat{\mathcal{S}^{k,L}_{\epsilon_2}}, \bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i\right) \leq d_0\left(\widehat{\mathcal{S}^{k,L}_{\epsilon_2}}, \bigcup_{i=1}^{p_0}[\mathsf{z}]^{k,L}_i\right)$. Comme par hypothèse $d_0\left(\widehat{\mathcal{S}^{k,L}_{\epsilon_2}}, \bigcup_{i=1}^{p_0}[\mathsf{z}]^{k,L}_i\right) \leq \eta/2$, on a $d_0\left(\widehat{\mathcal{S}^{k,L}_{\epsilon_2}}, \bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i\right) \leq \eta/2$. De même, on montre que $d_0\left(\mathcal{S}^{k,R}_{\epsilon_2}, \bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i\right) \leq \eta/2$. Par conséquent,

$$d_0\left(\widehat{\mathcal{S}^k_{\epsilon_2}}, \bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i\right) = \max\left(d_0\left(\widehat{\mathcal{S}^{k,L}_{\epsilon_2}}, \bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i\right), d_0\left(\mathcal{S}^{k,R}_{\epsilon_2}, \bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i\right)\right) \leq \eta/2. \tag{D.3}$$

De même, $d_0\left(\bigcup_{i=1}^{p_0}[\mathsf{z}]^{k,L}_i, \widehat{\mathcal{S}^k_{\epsilon_2}}\right) \leq \eta/2$, $d_0\left(\bigcup_{i=1}^{p_0}[\mathsf{z}]^{k,R}_i, \widehat{\mathcal{S}^k_{\epsilon_2}}\right) \leq \eta/2$ et, comme $[\mathsf{z}]^k_i \subset [\mathsf{s}]^k$ et $[\mathsf{z}]^{k,L}_i \cup [\mathsf{z}]^{k,R}_i = [\mathsf{z}]^k_i$,

$$d_0\left(\bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i, \widehat{\mathcal{S}^k_{\epsilon_2}}\right) = \max\left(d_0\left(\bigcup_{i=1}^{p_0}[\mathsf{z}]^{k,L}_i, \widehat{\mathcal{S}^k_{\epsilon_2}}\right), d_0\left(\bigcup_{i=1}^{p_0}[\mathsf{z}]^{k,R}_i, \widehat{\mathcal{S}^k_{\epsilon_2}}\right)\right) \leq \eta/2. \tag{D.4}$$

De $(D.3)$ et $(D.4)$, on déduit que $d\left(\bigcup_{i=1}^{p_0}[\mathsf{z}]^k_i, \widehat{\mathcal{S}^k_{\epsilon_2}}\right) \leq \eta/2$.

Nous avons démontré que la propriété 6 est vraie lorque BuildSp renvoit une feuille du sous-pavage image et reste vraie pour tous les appels antérieurs associés à des nœuds. Cette propriété est donc également vraie pour la racine de l'arbre image correspondant à $[\mathsf{s}_0]$. Par conséquent,

$$d\left(\mathsf{BuildSp}\left(\mathcal{L}^0_{\epsilon_1},\ [\mathsf{s}]^0,\ \epsilon_2\right),\ \bigcup_{i=1}^{p_0}\mathsf{f}_{[]}\left([\mathsf{x}]_i\right)\right)\ \mathsf{6}\ \eta/2.\ \blacksquare$$

Il suffit maintenant de combiner les deux parties de la démonstration

Preuve de la proposition 6 (conclusion) : Soit $\epsilon = \inf(\epsilon_1, \epsilon_2)$. D'après la première partie de la preuve,

$$d\left(\mathsf{f}\left(\widehat{\mathcal{X}}\right), \bigcup_{i=1}^{p_0}\mathsf{f}_{[]}\left([\mathsf{x}]_i\right)\right)\ \mathsf{6}\ \eta/2,$$

et d'après la seconde,

$$d\left(\mathsf{BuildSp}\left(\mathcal{L}^0_{\epsilon_1},\ [\mathsf{s}]^0,\ \epsilon\right),\ \bigcup_{i=1}^{p_0}\mathsf{f}_{[]}\left([\mathsf{x}]_i\right)\right)\ \mathsf{6}\ \eta/2.$$

Comme ImageSp renvoit le sous-pavage évalué par BuildSp, on a $\widehat{\mathcal{S}}_\epsilon = \mathsf{BuildSp}(\mathcal{L}^0_\epsilon,\ [\mathsf{s}],\ \epsilon)$. Par conséquent, $\mathsf{ImageSp}\left([\mathsf{s}],\ \mathsf{f}_{[]},\ \widehat{\mathcal{X}},\ \epsilon\right) = \widehat{\mathcal{S}}_\epsilon$. En appliquant l'inégalité triangulaire, il vient enfin

$$d\left(\mathsf{f}\left(\widehat{\mathcal{X}}\right),\ \widehat{\mathcal{S}}_\epsilon\right) = d\left(\mathsf{f}\left(\widehat{\mathcal{X}}\right), \mathsf{BuildSp}\left(\mathcal{L}^0_{\epsilon_1},\ [\mathsf{s}]^0,\ \epsilon\right)\right)\ \mathsf{6}\ \eta/2 + \eta/2 = \eta\ \blacksquare$$

Annexe E

Evolution de la balle

E.1 Description du mouvement de la balle

L'algorithme décrivant le mouvement de la balle en temps discret est le suivant, où l'accélération de la pesanteur est notée g, T est l'intervalle d'échantillonnage et r le rayon de la balle.

$\boxed{\mathsf{f}(x_k, \dot{x}_k)}$

const g, T, r;

$x_{k+} = x_k + \dot{x}_k T - g\frac{T^2}{2}$;

if $(x_{k+} > r)$

  return $\begin{pmatrix} x_{k+} \\ \dot{x}_k - gT \end{pmatrix}$;

else

  $\dot{x}_b = \sqrt{2g\,(x_k - r) + (\dot{x}_k)^2}$;

  $T_b = \dfrac{1}{g}\,(\dot{x}_k + \dot{x}_b)\,;$

  return $\begin{pmatrix} r + \dot{x}_b\,(T - T_b) - g\frac{(T-T_b)^2}{2} \\ \dot{x}_b - g\,(T - T_b) \end{pmatrix}$;

La hauteur de la balle à l'instant $(k+1)\,T$ est évaluée. Si cette hauteur est supérieure à r, la balle n'a pas encore heurté le sol. Sinon, la vitesse au rebond $\dot{x}_b$ ainsi que l'instant de rebond T_b sont évalués afin de calculer vitesse et hauteur à l'instant $(k+1)\,T$.

E.2 Fonction d'inclusion pour le mouvement de la balle

Un pendant intervalle de l'algorithme précédent s'obtient en remplaçant les occurrences des variables réelles par les variables intervalles et en prenant quelques précautions lors des tests.

$\boxed{\mathsf{f}_{[]}([x_k],[\dot{x}_k])}$

```
const g, T, r;
```

$[x_{k+}] = [x_k] + [\dot{x}_k]\,T - g\frac{T^2}{2};$

```
if (x_{k+} > r)
```

```
   return ( [x_{k+}]          );
          ( [ẋ_k] − gT        )
```

$[\dot{x}_b] = \sqrt{\left(2g\,([x_k]-r) + ([\dot{x}_k])^2\right)} \cap [0,+\infty[;$

$[T_b] = \frac{1}{g}\,([\dot{x}_k] + [\dot{x}_b]) \cap [0,+\infty[\,;$

```
if (x_{k+} ≤ r)
```

```
   return ( r + [ẋ_b]·(T − [T_b]) − g·(T − [T_b])^2/2 );
          ( [ẋ_b] − g·(T − [T_b])                     )
else
   return { ( [x_{k+}]          ) , ( r + [ẋ_b]·(T − [T_b]) − g·(T − [T_b])^2/2 ) }
          { ( [ẋ_k] − gT        )   ( [ẋ_b] − g·(T − [T_b])                     ) } ;
```

Les constantes sont les mêmes que dans l'algorithme ponctuel.

L'intervalle $[x_{k+}]$, encadrant la position de la balle à l'instant $(k+1)\,T$, est évalué en premier. Si la borne inférieure de $[x_{k+}]$ est supérieure à r, la balle n'a pas encore heurté le sol, sinon, il est *possible* qu'elle l'ait déjà fait. Des intervalles contenant l'instant de rebond $[T_b]$ et la vitesse de la balle $[\dot{x}_b]$ juste après le rebond sont évalués. Le fait que l'instant de rebond et que la vitesse sont positifs est pris en considération lors de leur évaluation.

Si la borne supérieure de $[x_{k+}]$ est plus petite que le rayon de la balle, la balle a certainement déjà heurté le sol. La position et la vitesse de la balle sont alors évalués en tenant compte de $[\dot{x}_b]$ et de $[T_b]$.

Lorsque $[x_{k+}]$ contient r, on ne peut pas savoir si la balle a ou non rebondi, et les deux situations doivent donc être prises en compte.

Index

Bibliographie

[Alefeld et al.83] Alefeld (G.) et Herzberger (J.). – *Introduction to Interval Computation.* – New York, Academic Press, 1983.

[Andrade et al.94] Andrade (M. V. A.), Comba (J. L. D.) et Stolfi (J.). – Affine arithmetic. *Interval'94.* – St Petersburg, 1994.

[Ansi85] A.N.S.I. (Institute of Electrical and Electronics Engeneers). – *A standard for binary floating-point arithmetic.* – New York, ANSI/IEEE Std. 754-1985, 1985.

[Ansi87] A.N.S.I. (Institute of Electrical and Electronics Engeneers). – *A standard for radix-independant floating-point arithmetic.* – New York, ANSI/IEEE Std. 854-1987, 1987.

[BB et al.99] Brahim-Belhouari (S.), Kieffer (M.), Fleury (G.), Jaulin (L.) et Walter (E.). – Model selection via worst-case criterion for nonlinear bounded-error estimation. – A paraître dans Proceedings 16th IEEE Instrumentation and Measurement Tech. Conf., Venise, 24-26 mai 1999.

[Beidler96] Beidler (J.). – *Data Structures and Algorithms.* – New York, Springer-Verlag, 1996.

[Berger79a] Berger (M.). – *Convexes et polytopes, polyèdres réguliers, aires et volumes.* – Volume 3. *Géométrie* [Berger87].

[Berger79b] Berger (M.). – *Espaces euclidiens, triangles, cercles et sphères.* – Volume 2. *Géométrie* [Berger87].

[Berger87] Berger (M.). – *Geometry I and II.* – Berlin, Springer-Verlag, 1987.

[Bertsekas et al.71] Bertsekas (D. P.) et Rhodes (I. B.). – Recursive state estimation for a set-membership description of uncertainty. *IEEE Trans. on Automatic Control*, vol. 16, 1971, pp. 117–128.

[Brumm et al.89] Brumm (P.) et Brumm (D.). – *Macro Assembler and Toolkit.* – Blue Ridge Summit, Tab Books Inc., 1989.

[Chen et al.97] Chen (G.), Wang (J.) et Shieh (L. S.). – Interval Kalman filtering. *IEEE Transaction on Aerospace and Electronic Systems*, vol. 33, no. 1, 1997, pp. 250–258.

[Crowley89] Crowley (J.). – World modeling and position estimation for a mobile robot using ultrasonic ranging. *Proc. IEEE International Conference on Robotics and Automation*, pp. 674–680. – Scottsdale, Arizona, 1989.

[Davoust et al.98] Davoust (M. E.), Fleury (G.) et Oksman (J.). – A parametric estimation approach for grooves dimensioning using remote field eddy current inspection. – 1998. Soumis à Research in Nondestructive Evaluation.

[Delannoy91] Delannoy (C.). – *C++ guide complet*. – Paris, Eyrolles, 1991.

[Didrit97] Didrit (O.). – *Analyse par intervalles pour l'automatique; résolution globale et garantie de problèmes non linéaires en robotique et commande robuste*. – Orsay, Thèse de doctorat, Université Paris-Sud, juin 1997.

[Drumheller87] Drumheller (M.). – Mobile robot localization using sonar. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 9, no. 2, 1987, pp. 325–332.

[Durieu et al.96a] Durieu (C.), Polyak (B.) et Walter (E.). – Ellipsoidal state outer-bounding for MIMO systems via analytical techniques. *Proc. IMACS—IEEE–SMC CESA'96 Symposium on Modelling and Simulation*, pp. 843–848. – Lille, 1996.

[Durieu et al.96b] Durieu (C.), Polyak (B.) et Walter (E.). – Trace versus determinant in ellipsoidal outer bounding with application to state estimation. *Proc. 13th IFAC World Congress*, pp. 43–48. – San Francisco, 1996.

[Gardenes et al.85] Gardenes (E.), Mielgo (H.) et Trepat (A.). – Modal intervals : reasons and ground semantics. *Interval Mathematics 1985*. – Berlin, 1985.

[Godfrey83] Godfrey (K.). – *Compartimental models and their application*. – London, Academic Press, 1983.

[Goldberg56] Goldberg (K.). – A matrix with real caracteristic roots. *J. Res. National Bureau of Standards*, no. 56, 1956, p. 87.

[Grimson et al.87] Grimson (W. E.) et Lozano-Pérez (T.). – Localizing overlapping parts by searching the interpretation tree. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 9, no. 4, 1987, pp. 469–482.

[Halbwachs et al.97] Halbwachs (E.) et Meizel (D.). – Multiple hypothesis management for mobile vehicule localization. *CD Rom of the European Control Conference*. – Louvain, 1997.

[Hammer et al.95] Hammer (R.), Hocks (M.), Kulish (U.) et Ratz (D.). – *C++ Toolbox for Verified Computing*. – Berlin, Springer Verlag, 1995.

[Han69] *Topics in Interval Analysis*, éd. par Hansen (E.). – London, Oxford University Press, 1969.

[Hansen et al.81] Hansen (E. R.) et Sengupta (S.). – Bounding solutions of systems of equations using interval arithmetic. *BIT*, vol. 21, 1981, pp. 203–211.

[Hansen92] Hansen (E. R.). – *Global Optimization using Interval Analysis*. – New York, Marcel Dekker, 1992.

[Hyb93] *Hybrid Systems. Lecture Notes in Computer Science, vol. 736*, éd. par Grossman (R. L.), Nerode (A.), Ravn (A. P.) et Rischel (H.). – New York, Springer Verlag, 1993.

[Jacquez72] Jacquez (J. A.). – *Compartmental analysis in biology and medecine*. – Amsterdam, Elsevier, 1972.

[Jaulin et al.93] Jaulin (L.) et Walter (E.). – Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, vol. 29, no. 4, 1993, pp. 1053–1064.

[Jaulin et al.96] Jaulin (L.), Walter (E.) et Didrit (O.). – Guaranteed robust nonlinear parameter bounding. *IMACS—IEEE–SMC CESA'96 Symposium on Modelling and Simulation*, pp. 1156–1161. – Lille, France, 1996.

[Jaulin et al.98] Jaulin (L.), Kieffer (M.), Walter (E.) et Meizel (D.). – Guaranteed robust nonlinear estimation with application to robot localization. – Soumis à *Automatica*, 1998.

[Kearfott95] Kearfott (R. B.). – A Fortran 90 environment for research and prototyping of enclosure algorithms for nonlinear and global optimization. *ACM Trans. Math. Software*, vol. 21, no. 1, 1995, pp. 63–78.

[Kieffer et al.98a] Kieffer (M.), Jaulin (L.) et Walter (E.). – Guaranteed recursive nonlinear state estimation using interval analysis. *Proc. 37th IEEE Conference on Decision and Control*, pp. 3966–3971. – Tampa, Florida, 16-18 décembre 1998.

[Kieffer et al.98b] Kieffer (M.), Jaulin (L.) et Walter (E.). – Guaranteed recursive nonlinear state estimation using interval analysis. – Soumis pour publication., 1998.

[Kieffer et al.98c] Kieffer (M.) et Walter (E.). – Interval analysis for guaranteed nonlinear parameter estimation. *MODA 5-Advances in Model-Oriented Data Analysis and Experiment Design*, éd. par Atkinson (A. C.), Pronzato (L.) et Wynn (H. P.). pp. 115–125. – Heidelberg, 1998.

[Kieffer et al.99a] Kieffer (M.), Jaulin (L.), Walter (E.) et Meizel (D.). – Robust autonomous robot localization using interval analysis. – 1999. A paraître dans *Reliable Computing*.

[Kieffer et al.99b] Kieffer (M.), Jaulin (L.), Walter (E.) et Meizel (D.). – Guaranteed mobile robot tracking estimation using interval analysis. *Proceedings of MISC'99 Workshop on Application of Interval Analysis to System and Control*. – Girone, 24-26 février 1999.

[Klatte et al.93] Klatte (R.), Kulisch (U.), Wieth (A.), Lawo (C.) et Rauch (M.). – C-xsc*: A C++ Class Library For Extended Scientific Computing*. – Berlin Heidelberg, Springer Verlag, 1993.

[Klinger76] Klinger (A.). – Experiment in picture representation using regular decomposition. *Comput. Graphics Image Proc.*, vol. 5, 1976, pp. 68–105.

[Lahanier et al.87] Lahanier (H.), Walter (E.) et Gomeni (R.). – OMNE: a new robust membership-set estimator for the parameters of nonlinear models. *J. of Pharmacokinetics and Biopharmacetics*, vol. 15, 1987, pp. 203–219.

[Leonard et al.91] Leonard (J. J.) et Durrant-Whyte (H. F.). – Mobile robot localization by tracking geometric beacons. *IEEE Trans. on Robotics and Automation*, vol. 7, no. 3, 1991, pp. 376–382.

[Leonard et al.92] Leonard (J. J.) et Durrant-Whyte (H. F.). – *Directed Sonar Sensing for Mobile Robot Navigation*. – Boston, Kluwer Academic Publishers, 1992.

[Leveque et al.97] Lévêque (O.), Jaulin (L.), Meizel (D.) et Walter (E.). – Vehicule localization from inaccurate telemetric data: a set inversion approach. *Proceedings of the 5th IFAC Symposium on Robot Control SY.RO.CO.'97*, pp. 179–186. – Nantes, France, 1997.

[Leveque98] Lévêque (O.). – *Méthodes ensemblistes pour la localisation de véhicules*. – Compiègne, Thèse de doctorat, Université de Technologie, 1998.

[Lippman89] Lippman (S. B.). – *C++ Primer*. – Reading, Massachusetts, Addison-Wesley, 1989.

[Mackintosh et al.96] Mackintosh (D. D.), Atherton (D. L.), Schmidt (T. R.) et Russell (D. E.). – Remote field eddy current for examination of ferromagnetic tubes. *Materials Evaluation*, vol. 54, 1996, pp. 652–657.

[Maksarov et al.96] Maksarov (D.) et Norton (J. P.). – State bounding with ellipsoidal set description of the uncertainty. *Int. J. of Control*, vol. 65, no. 5, 1996, pp. 847–866.

[Mckinnon et al.96] McKinnon (K.), Millar (C.) et Mongeau (M.). – Global optimization for the chemical and phase equilibrium problem using interval analysis. *State of the Art in*

Global Optimization, éd. par Floudas (C.) et Parlados (P.), pp. 365–382. – Kluwer Academic Press, 1996.

[Metairie et al.90] Metairie (C.) et Polian (N.). – *C++ Utilisation d'un langage orienté objet.* – Paris, Eyrolles, 1990.

[Moore66] Moore (R. E.). – *Interval Analysis.* – Englewood Cliffs, New Jersey, Prentice-Hall, 1966.

[Moore79] Moore (R. E.). – *Methods and Applications of Interval Analysis.* – Philadelphia, Pennsylvania, SIAM Publ., 1979.

[Moore92] Moore (R. E.). – Parameter sets for bounded-error data. *Mathematics and Computers in Simulation*, vol. 34, 1992, pp. 113–119.

[Neira et al.96] Neira (J.), Horn (J.), Tardoz (J. D.) et Schmidt (G.). – Multisensor mobile robot localization. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 673–679. – Mineapolis, USA, 1996.

[Neumaier90] Neumaier (A.). – *Interval Methods for Systems of Equations.* – London, Cambridge University Press, 1990.

[Nor94] Special issue on bounded-error estimation: Issue 1. *Int. J. Of Adaptive Control and Signal Processing,* vol. 8, no. 1, éd. par Norton (J. P.), pp. 1–118. – 1994.

[Nor95] Special issue on bounded-error estimation: Issue 2. *Int. J. Of Adaptive Control and Signal Processing,* vol. 9, no. 1, éd. par Norton (J. P.), pp. 1–132. – 1995.

[Pichat et al.93] Pichat (M.) et Vignes (J.). – *Ingénierie du contrôle de la précision des calculs sur ordinateur.* – Paris, Éditions Technip, 1993.

[Ple96] *Bounding Approaches to System Identification*, éd. par Milanese (M.), Norton (J.), Piet-Lahanier (H.) et Walter (E.). – New York, Plenum Press, 1996.

[Pronzato et al.96] Pronzato (L.) et Walter (E.). – Robustness to outliers of bounded-error estimators and consequences on experiment design. *Bounding Approaches to System Identification*, éd. par Milanese (M.), Norton (J.), Piet-Lahanier (H.) et Walter (E.). pp. 199–212. – New York, USA, 1996.

[Rall81] Rall (L. B.) (édité par). – *Automatic Differentiation: Techniques and Application.* – Berlin, Springer Verlag, 1981, *Lecture Notes in Computer Science*, volume 120.

[Ratschek et al.88] Ratschek (H.) et Rokne (J.). – *New Computer Methods for Global Optimization.* – New York, Wiley, 1988.

[Schweppe68] Schweppe (F. C.). – Recursive state estimation: Unknown but bounded errors and system inputs. *IEEE Transactions on Automatic Control*, vol. 13, no. 1, 1968, pp. 22–28.

[Schweppe73] Schweppe (F. C.). – *Uncertain Dynamic Systems*. – Englewood Cliffs, NJ, Prentice-Hall, 1973.

[Vehi98] Vehi (J.). – *Analysis and Design of Robust Controllers by means of Modal Intervals*. – Spain, Thèse de doctorat, Universitat de Girona, 1998.

[Wal90] Special issue on parameter identifications with error bounds. *Mathematics and Computers in Simulation,* vol. 32, no. 5&6, éd. par Walter (E.), pp. 447–607. – 1990.

[Walter et al.88] Walter (E.) et Piet-Lahanier (H.). – Estimation of the parameter uncertainty resulting from bounded-error data. *Mathematical Biosciences*, vol. 92, 1988, pp. 55–74.

[Walter et al.97] Walter (E.) et Pronzato (L.). – *Identification of Parametric Models from Experimental Data*. – London, Springer-Verlag, 1997.

[Walter82] Walter (E.). – *Identifiability of state space models*. – Berlin, Springer-Verlag, 1982.